# DETECTING THE SPIN-GLASS STATE THROUGH NEURAL NETWORKS

An Undergraduate Research Scholars Thesis

by

HUMBERTO MUÑOZ BAUZA

Submitted to the Undergraduate Research Scholars program
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by Research Advisor: Dr. Helmut G. Katzgraber

May 2017

Major: Physics and Mathematics

## TABLE OF CONTENTS

## ABSTRACT

Detecting the Spin-Glass State through Neural Networks



Humberto Muñoz Bauza
Department of Physics and Astronomy
Texas A&M University

Research Advisor: Dr. Helmut G. Katzgraber
Department of Physics and Astronomy
Texas A&M University



Neural networks, a class of artificial intelligence techniques that extract patterns from data and are suitable for tasks such as function approximation and image classification [1, 2], are finding applications in computational condensed matter physics, where simulation data can be reliably generated in large quantities. For example, they can classify phases in classical and quantum lattice models using Monte Carlo simulation data [3, 4]. In this application, one of their useful characteristics is their *generalization* capacity: their ability to identify a similar phase transition when the model Hamiltonian is altered. A controversial point in the theory of spin glasses, systems characterized by frustration and a complex energy landscape, is the existence of a spin-glass state in the presence of random fields. However, a Monte Carlo study by Young and Katzgraber [5] has ruled this out for the 3D Edwards-Anderson model with Gaussian interactions. This suggests a excellent test for the generalization capacity of a neural network trained to identify the spin-glass state. In this work, a 3D convolutional neural network was implemented and taught to recognize the spin-glass state of the 3D Edwards-Anderson model using data from Monte Carlo simulations. The inference results collected are found to be consistent with the absence of a spin-glass state in a field. The strengths and weaknesses of the phase classifier are evaluated and further directions are discussed.

## ACKNOWLEDGMENTS

I highly thank Dr. Helmut Katzgraber for his constant guidance during the course of this project. The computation time for this project was available through the use of Texas A&M High Performance Research Computing resources. I additionally thank Firas Hamze (D-Wave Systems), Chao Fang, and Dr. Ehsan Khatami (San José State U., CA) for constructive discussions, and Andrew Ochoa for proofreading the initial manuscript. My studies and activities in the Department of Physics and Astronomy had the support of Sherree Kessler, Dr. Alexey Belyanin, and many other members of the staff and faculty of the Department. This last year also had the generous financial support of the Marianne '76 and Robert '77 Hamm Endowed Scholarship. I could not have asked for better personal support from my family, colleagues, classmates, and Deanna.

## NOTATION AND NOMENCLATURE

| | |
|---|---|
| ML | Machine Learning |
| NN | Neural Networks |
| CNN | Convolutional Neural Network |
| (3D-)EA | (Three-dimensional) Edwards-Anderson |
| AT | Almeida-Thouless (line) |
| SG | Spin Glass (state/model) |
| PM | Paramagnetic |
| FM | Ferromagnetic |
| AFM | Antiferromagnetic |
| SLP | Single Layer Perceptron |
| MLP | Multi-layer Perceptron |
| ReLU | Rectified Linear Unit |
| $a, b, c, \ldots$ | Scalars and vectors, and some matrices |
| $\vec{a}, \vec{b}, \vec{c}$ | Vectors/microstates |
| $A, B, C \ldots$ | Matrices/Rank-2 tensors or high-rank tensors |
| $\langle \cdot \rangle$ | Thermal average/expectation value |
| $[\cdot]$ | Instance average |

# CHAPTER 1

# INTRODUCTION

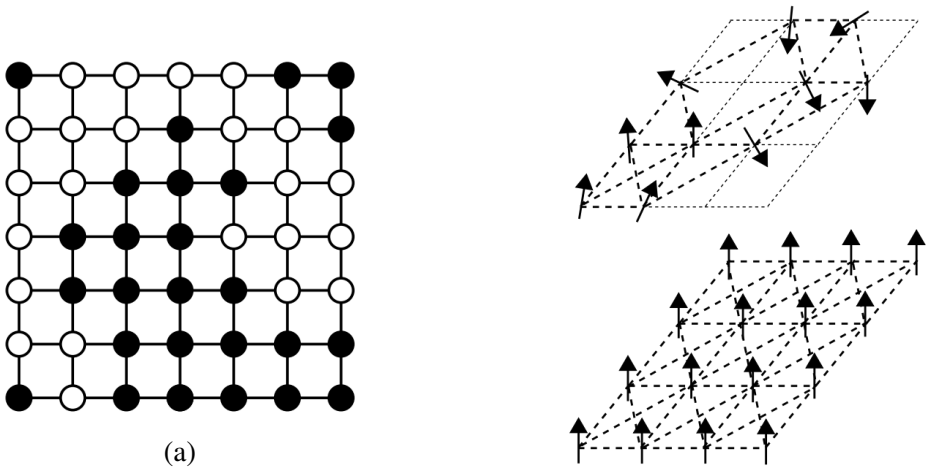
## 1.1 The Ising Model and Spin Glasses

A material with a highly ordered, periodic spatial structure is called a crystal. Various familiar compounds, such as minerals, metals, insulators, and semiconductors are examples of crystalline materials. When ions in a crystal have unpaired electrons, this creates a magnetic dipole on the ion site due to the extra electron's spin. In the presence of an external magnetic field, when the spins tend to align along the direction of the field, the material is said to be *paramagnetic* (PM). If additionally, the spins tend to align in the same direction as their nearest neighbors through electron interactions, the material is said to be *ferromagnetic* (FM). The opposite case, where a spin will tend to be aligned against the direction of its neighbors, describes an *antiferromagnetic* (AFM) material. [6]

A glass, unlike a crystal, is amorphous: its spatial structure has no microscopic order or periodicity, so the atoms have spatial disorder. Specifically, the disorder is frozen into the glass, unlike a molten material. Analogously, if the magnetic structure of a material is frozen in and has no microscopic order, the material is said to be in a *spin-glass* (SG) state, and is referred to as a spin glass. The spin-glass state is characterized by the random and competing interactions between spins, exhibiting a mixture of ferromagnetic and antiferromagnetic interactions [7]. Spin glasses thus have a large number of metastable states, and thus are referred to as having a "complex energy landscape."

### *1.1.1 The Ising Model of Ferromagnetism*

Ferromagnetism is characterized by a simple type of lattice model called the *Ising model*. In the two-dimensional version of the model, we have a 2D square lattice with



(a) (b)

Figure 1.1: (a) An illustration of the spins and interactions of the Ising model on a 2D square lattice. (Periodic boundary conditions not depicted.) (b) Qualitative distinction between an ordered and a glassy magnetic structure.

$N = L^2$ sites. Each site $i$ has an associated spin $s_i = \pm 1$ along a single axis only (e.g., the $z$ axis). Each spin has an interaction bond with its four nearest neighbors,[1] and every bond has the same ferromagnetic interaction strength $J$. That is, the internal energy of two interacting spins in the same direction is $-J$, and that of two spins in opposite directions is $+J$. An external field $h$ is also applied along the axis of the spins and contributes an energy of $-hs_i$ on each spin. The Ising model Hamiltonian (the total energy of the system) is thus written as

$$H = -J \sum_{\langle i\, j \rangle} s_i s_j - h \sum_i s_i, \tag{1-1}$$

where the sum $\sum_{\langle i\, j \rangle}$ is a sum over all *pairs* of interacting spins (no edge duplication).

There is also the 1D Ising model, which is a simple chain of interacting spins, and the 3D Ising model, which is a cubic lattice with 6 interacting neighbors per spin. In dimensions 2 and higher, the Ising model exhibits a ferromagnetic state below a critical

[1] This is also true for spins on the boundaries if we assume *periodic* boundary conditions.

temperature[2] $T_c > 0$, making it a simple yet successful model for ferromagnetic materials. The one-dimensional model was first analyzed exactly by Ernst Ising [8] in 1925, hence the model's name, while the two-dimensional model was later analyzed by Lars Onsager [9].

### *1.1.2 The Edwards-Anderson Model of Spin Glasses*

The *Edwards-Anderson* (EA) model [10], like the Ising model, is a lattice model on a $N = L^2$, or $L^3$, etc... square lattice. However, the spin interactions and fields are no longer uniform. Each interaction and field is now randomly and independently drawn from some appropriate probability distributions $P_J$ and $P_h$. The Hamiltonian has the more general form

$$H = -\sum_{\langle i\, j\rangle} J_{ij} s_i s_j - \sum_i h_i s_i, \tag{1-2}$$

where the $J_{ij}$ are now random variables drawn from $P_J$ and the $h_i$ are random variables drawn from $P_h$.

To exhibit frustration, $P_J$ should have a sufficiently large probability of AFM interactions $J_{ij} < 0$. Typical probability distribution choices are a Gaussian (normal) distribution $P_J = N(J_0, \tilde{J}^2)$ with mean $J_0$ and variance $\tilde{J}^2$, or a bimodal distribution $P_J = B_p(J)$ where $+J$ is selected with probability $p$ and $-J$ with probability $1 - p$. Typically, the probability distribution is symmetric, so $J_0 = 0$ in the Gaussian EA model and $p = 0.5$ in the bimodal EA model. The random fields are also optional, and in the zero field EA model they can all be set to 0. In the non-zero field model, they are drawn from a normal distribution with mean 0 and variance $H$. Hereon, the energy units for the EA model are also chosen so that $\tilde{J} = 1$ in the Gaussian model (or $J = 1$ in the bimodal model).

Not all variations of the EA model will exhibit a spin-glass state. The 2D model does not have a spin-glass state below a non-zero temperature. Also, in the Gaussian 3D-EA

[2]The critical temperature $T_c$ is also called the *Curie temperature* of the ferromagnet.

model, the spin-glass state appears to vanish in the non-zero field model, despite being present in the zero field model [11].

### *1.1.3 Statistical Physics and Critical Behavior*

Statistical physics allows us to describe the properties of systems such as the Ising Model in conditions of *thermodynamic equilibrium* [12]. If a system is in equilibrium held at a temperature $T$, then the probability that the system is in a particular arrangement of spins $\vec{s}$ at any given point in time is given by the *Boltzmann distribution* of the system at the temperature $T$

$$P(\vec{s}) = \frac{1}{Z} e^{-H(\vec{s})/kT}, \tag{1-3}$$

where $H(\vec{s})$ is the energy of the spin configuration, $k$ is the Boltzmann constant, and $Z$ is the normalization factor

$$Z = \sum_{\vec{s}} e^{-H(\vec{s})/kT}. \tag{1-4}$$

$Z$ is also called the *partition function*. Each allowed spin configuration $\vec{s}$ is referred to as a *microstate*. Using the Boltzmann distribution, it is possible to calculate averages of physical quantities (also called *observables*) in some simple cases. For example, the internal energy of the system is the average of the Hamiltonian over the Boltzmann distribution,

$$E = \langle H \rangle = \sum_{\vec{s}} H(\vec{s}) P(\vec{s}) = \frac{1}{Z} \sum_{\vec{s}} H(\vec{s}) e^{-H(\vec{s})/kT}. \tag{1-5}$$

It is useful to note that $E = kT^2 \frac{\partial Z}{\partial T}$. The fluctuation of the internal energy is quantified by the variance $\Delta E^2$ of the Hamiltonian over the Boltzmann distribution, i.e.

$$\Delta E^2 = \left\langle H^2 \right\rangle - \langle H \rangle^2 . \tag{1-6}$$

Deriving (1-5) with respect to $T$ in fact gives

$$\frac{\partial E}{\partial T} = \frac{1}{kT^2}\left(\left\langle H^2\right\rangle - \left\langle H\right\rangle^2\right). \tag{1-7}$$

Thus, the thermodynamic *heat capacity* (per spin) is related to the thermal fluctuation by $c_V = \frac{1}{N}\frac{\partial E}{\partial T} = \Delta E^2/NkT^2$.

Statistical mechanics also provides many theoretical tools for analyzing *phase transitions*, which are marked by singularities in certain observables and represent a boundary between highly distinct behaviors in the same system. Of importance in phase transition theory is an *order parameter*, a thermodynamic observable that can be used to find a "signal" for a phase transition, if one exists. In the Ising model, the most natural order parameter is the average magnetization, which quantifies the magnetic order of the spins,

$$m = \frac{1}{N}\sum_i s_i = M/N, \tag{1-8}$$

where $M$ is the net magnetization. At high temperatures, each spin tends to overcome the ferromagnetic interactions with its neighbors, and thus tend to take on $\pm 1$ at random and independently. Hence, $m \approx 0$ and the system is in a paramagnetic state. However, at low enough temperatures, low energy states become highly favored. Thus, all spins tend to arrange themselves in the same direction, and so $m \approx \pm 1$, which represents the ferromagnetic state. The fluctuation of the magnetization is related to the *magnetic susceptibility* (per spin)

$$\chi = \frac{1}{N}\frac{\partial \left\langle M\right\rangle}{\partial h} = \frac{1}{NkT}\Delta M^2 = \frac{N}{kT}\left(\left\langle m^2\right\rangle - \left\langle m\right\rangle^2\right). \tag{1-9}$$

The characteristics of a system very close to a phase transition form its *critical behavior*, and are quantified by *critical exponents* that describe power-law divergent observables. In the Ising model, $c_V$ and $\chi$ diverge in the vicinity of the phase transition temperature $T_c$

between the FM and PM phases. This is an example of critical behavior found in lattice models with a phase transition. Specifically, if $t = |T - T_c|$, then

$$\chi \propto t^{-\gamma}$$
$$C_V \propto t^{-\alpha},$$

where $\gamma$ and $\alpha$ are two critical exponents characterizing the Ising model. According to Onsager's solution, $\gamma = 7/4$ for the 2D Ising model. However, $\alpha = 0$ because the specific heat diverges logarithmically, which is slower than any power-law divergence.

For finite size-systems a true divergence is not possible, but the behavior of the resulting peaks as the system size is varied can be obtained from simulation and analyzed to infer some critical properties (*finite-size scaling*) [13]. Certain unitless observables are also helpful for analyzing critical behavior in finite systems, since they have the convenient property of being independent of the system size right *at* the critical temperature (within reasonably large system sizes). One example for the Ising model is the Binder ratio [14]

$$g = \frac{1}{2}\left(3 - \frac{\langle m^4 \rangle}{\langle m^2 \rangle^2}\right). \tag{1-10}$$

### *1.1.4 Statistical Physics of Spin Glasses*

The critical properties of spin glasses, unlike the Ising model, cannot be described with $m$ as the order parameter, since by definition the spins are disordered and uncorrelated. Instead, the appropriate order parameter for a spin glass needs should take into account the presence of many low temperature metastable states. Additionally, calculating thermodynamic observables for a spin glass encounters the issue that the parameters of the Hamiltonian are random variables. Due to these issues, spin glasses pose many theoretical, experimental, and computational challenges that are described in a large body of

literature. Some relevant starting points beyond the summary given here may be found in Refs. [7, 15, 16].

Parameter samples of the Hamiltonian of a model with random variables (e.g. the EA model) are called *instances* of the model. To calculate an observable for a spin glass, in addition to averaging over the Boltzmann distribution for a particular instance, it is necessary to then average over instances (i.e. averaging over the parameter probability distribution of the spin glass Hamiltonian). For example, the average energy observable is written as

$$E = \sum_{\langle i\, j\rangle} \left[J_{ij} \left\langle s_i s_j \right\rangle\right] \tag{1-11}$$

where $\langle s_i s_j \rangle$ is the thermal spin correlation for a particular instance, and $[\cdot]$ is the instance average over the bond distribution of $J_{ij}$.

The spin glass order parameter should qualitatively be similar to the magnetization: 0 at high temperatures (disordered state) and $\pm 1$ at low temperatures (spin-glass state. Consider the fates of two different systems with the same Hamiltonian instance (called *replicas* of the instance) as they enter the spin-glass state. They will settle on an energy valley independently of each other, but in accord with the Boltzmann distribution. The *spin overlap parameter* between the two replicas is defined by

$$q_{ab} = \sum_i s_i^{(a)} s_i^{(b)}, \tag{1-12}$$

where $(a)$ and $(b)$ are the indices of the replicas at the same temperature (and fields if present). If the energy valleys of the replicas coincide, $q_{ab} \approx 1$. If the replicas are completely anticorrelated, $q_{ab} \approx -1$. (In zero field models, where spin flip symmetry holds, energy valleys with $s_i \to -s_i$ are also equally present.) If the overlap is measured at high temperatures, the spins become uncorrelated, so $q_{ab} \approx 0$. At low temperatures, intermedi-

ate $q_{ab}$ indicate energy valleys with partial coincidence. Thus, one appropriate spin glass order parameter $q$ is the thermodynamic and instance averaging of the overlap measurements $q_{ab}$. That is,

$$q = \sum_i \left[ \left\langle s_i^{(a)} s_i^{(b)} \right\rangle \right]. \tag{1-13}$$

One useful observable is the spin glass wave-vector susceptibility

$$\chi_{\vec{k}}^{(SG)} = \frac{1}{N} \sum_{ij} \left[ \left( \langle s_i s_j \rangle - \langle s_i \rangle \langle s_j \rangle \right)^2 \right] e^{i\vec{k}\cdot\vec{R}_{ij}} \tag{1-14}$$

where $\vec{R}_{ij}$ is the separation vector between the sites $i$ and $j$, and the double sum is over all pairs of sites. In terms of the susceptibility, we can define the *correlation length* divided by the system length, a unitless observable, as

$$\xi_L / L = \frac{1}{2\sin(k_{min}/2)} \left( \frac{\chi_0^{(SG)}}{\chi_{\vec{k}_{min}}^{(SG)}} - 1 \right)^{1/2}, \tag{1-15}$$

where $\vec{k}_{min} = 2\pi/L\hat{x}$ is a minimal wave vector for the lattice. The correlation length, like the Binder ratio for the Ising model, can be collected in Monte Carlo simulations and observed over varying system sizes to find a spin glass transition. This was the procedure done in Ref. [11] to attempt a detection of a non-zero field phase transition in the 3D-EA model.

Some theoretical models for spin glasses like the EA model exist. The *droplet theory* [17] describes metastable states as resulting from excitations of "droplets" of spins. The *replica symmetry breaking* [18] model (RSB) postulates that the behavior of systems with short-ranged interactions (e.g. nearest neighbor interactions as in the EA model) is similar to spin glass models where the interactions are all-to-all (such as the *Sherrington-Kirkpatrick* model [19]).

### *1.1.5 The Almeida-Thouless Line*

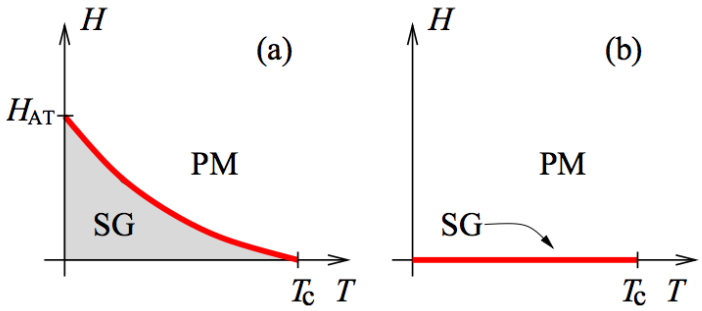



Figure 1.2: (a) Phase diagram with a spin-glass state present at non-zero fields. The hypothetical boundary of such a spin glass phase is the AT line. The field at which the transition exists at $T = 0$ is denoted by $H_{AT}$. (b) Phase diagram with the spin-glass state only at zero field. [11]

The existence of a spin-glass state in a field for realistic, short-ranged models is an open question. If such a state exists, the transition between the spin-glass state and the paramagnetic state in non-zero fields is called an *Almeida-Thouless* (AT) line. The droplet picture predicts that there is no AT line, while RSB predicts that an AT line does exist [11]. These distinct possibilities are illustrated as phase diagrams in figure 1.2.

Some previous work appears to suggest that the EA model does not have an AT line. In Ref. [11], Young and Katzgraber could not find evidence of a spin glass transition at non-zero fields in the Gaussian EA model. By simulating a diluted bimodal EA model through a diagonal path in the H-T phase space, Jörg *et. al.* (Ref. [20]) also failed to find an AT line.

## 1.2 Neural Networks and Deep Learning

Inspired by analogy with neurobiology, a *neural network* is a machine, physical or simulated, that models the brain in performing a particular task, such as data classification and function approximation/interpolation [2]. As information-processing systems, they exhibit the complex, nonlinear, and parallel functionality of the human brain at a smaller scale. Neural networks may successfully perform their intended tasks after a *learning process* that adjusts their internal parameters as they go through training data. We say that the neural networks are *trained* through the use of a learning algorithm on the training data set. Training algorithms can be classified into supervised and unsupervised algorithms. Supervised learning can be used to perform input/output associations, while unsupervised learning attempts to build the neural network's knowledge representation through data input only [1, 2].

The idea of using neural networks for computation was introduced by McCulloch and Pitts in 1943 [21]. The simplest model for a neural network trained through supervised learning is the *single layer perceptron* (SLP), introduced by Rosenblatt in 1958 [22]. The SLP can be successfully used as a binary classifier for data that is linearly separable, i.e., there is a hyperplane in $\mathbb{R}^n$ that separates the two classes of the data, as illustrated in figure 1.3.

Modern neural network research has seen the rise of *deep learning* techniques, in which the knowledge capacity of a neural network is powered by a "deep" architecture, i.e. a large number of layers and connections, which allows the neural network to systematically extract features and patterns from inputs in a simple-to-complex progression [1]. When dealing with inputs such as images, a logical simplification on the layers that reduces the amount of resources needed is to use a *convolutional layer*, described in Sec. 1.2.5. Thus, *deep convolutional neural networks*, based on the use of many such layers,

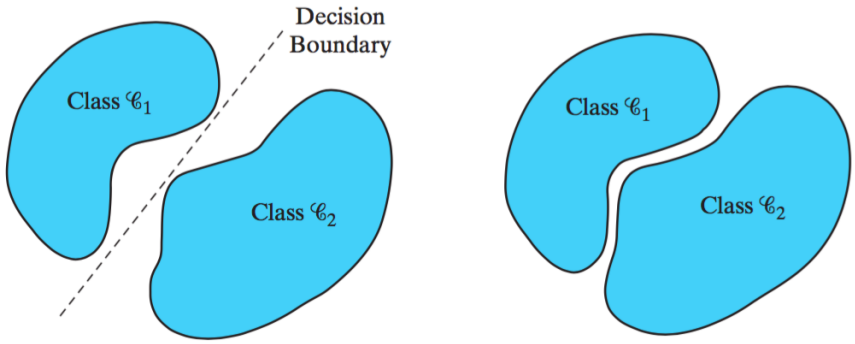



Figure 1.3: An illustration of linear separability and non-separability in $\mathbb{R}^2$ for training data for an SLP [2].

have been very successful in tasks such as image classification. The "AlexNet" architecture [23] pioneered the technique for the highly accurate classification of 1.2 million images representing about 1000 different everyday objects and animals.

This section develops some of the background theory for feedforward neural networks and specifically convolutional neural networks, and is followed by an outline of the gradient descent learning algorithm, which performs *supervised learning* on a neural network by adjusting the parameters to minimize a loss function representing the error in the output.

### *1.2.1 The Neuron Model*

A "neuron" is the fundamental information-processing unit of a neural network. The paths of the input signals $x_i$ are called *synapses*. Each synapse is associated with a weight $w_i$, which multiplies the input signal going into the neuron. Additionally, one synapse, called a *bias*, can propagate a signal $b$ that is independent of the input signals. The neuron then acts as an adder on all the weighed input signals. That is, its output is

$$z = \sum_i w_i x_i + b. \tag{1-16}$$

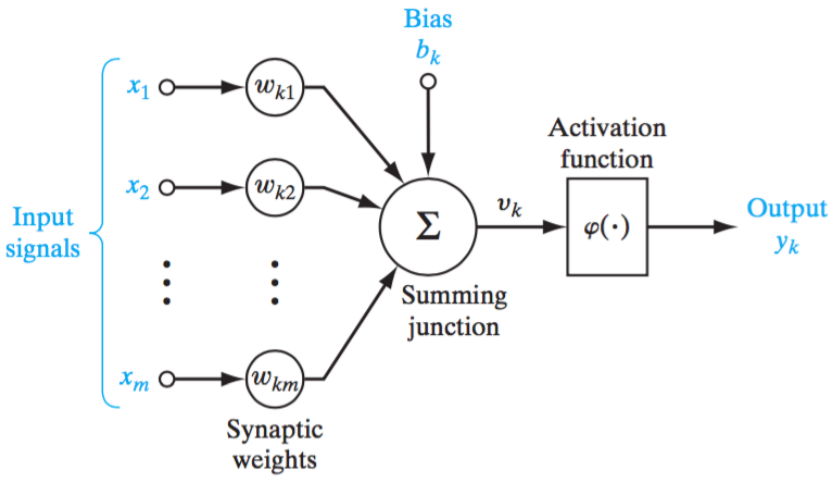



Figure 1.4: Diagram of the $k$th neuron in a multineuron perceptron , illustrating an affine transformation on the input $x$ followed by the activation function $\phi$. [2]

The neuron output is then passed through a *activation function*, which serves as a "squasher" limiting the range of the synaptic output and acts as a nonlinear element. Thus, if $\phi$ is the activation function the synaptic output of the neuron is given by

$$a = \phi(z). \tag{1-17}$$

A traditional activation function is the *sigmoid*

$$\phi(z) = \sigma(z) = \frac{1}{1 + e^{-z}}, \tag{1-18}$$

which limits the synaptic output to the range $(-1, +1)$. An activation function that is more popular in deep learning has a simpler form

$$\phi(z) = \max(0, z). \tag{1-19}$$

This is called the ReLU (Rectified Linear Unit) and it has the desirable property that its derivative is large for many in comparison with the sigmoid. Thus, it helps accelerate learning in the gradient descent algorithm described in the next section.

### *1.2.2 The Single Layer Perceptron*

The single layer perceptron is the simplest model of a neural network that can perform a binary classification task on a linearly separable data set. It is defined as consisting of the following components

1. A layer of $m$ signals that simply propagate input data as a vector in $\mathbb{R}^m$. This is called the *input layer*.

2. A single *output layer* consisting of one neuron, whose synaptic inputs are the $m$ signals from the input layer, and is parametrized by a vector of $m$ synaptic weights $w_i$ and a bias scalar $b$. The weights and biases are adjustable parameters to be tuned during learning.

3. The activation function is simply the sign function, so the output is $y = \mathrm{sgn}(z)$, which is $1$ if $z > 0$, and $-1$ otherwise.

Suppose we have a set of $n$ inputs $\vec{x}_{1,\dots,n}$, and a set of $n$ desired outputs $d_{1,\dots,n} = \pm 1$ which serve to label each input as one of two possible classes. Then we can furnish the SLP with a learning algorithm given by Algorithm 1 which iteratively updates the parameters in accordance with the *output error* $e = y - d$. It can be shown that if the inputs are classified in a linearly separable way, then the perceptron is guaranteed to converge to a set of parameters that will correctly classify all of the inputs. This is called the *Perceptron Convergence Theorem* [2]. The algorithm depends on a parameter $\eta$ called the *learning rate*. This is one of the most important parameters for a supervised learning algorithm, and determines how much stride the algorithm has when adjusting a parameter.

**Algorithm 1** Training algorithm for the single layer perceptron according to the perceptron convergence theorem.

**procedure** SLP-TRAIN( Initial parameters $\vec{w}, b$, sequence of $n$ inputs $\vec{x}_i$, positive learning rate $\eta$ )
    **for** $i = 1, 2, \ldots n$ **do**
        $z \to \vec{w} \cdot \vec{x}_i + b$
        $y \to \text{sgn}(z)$
        Calculate output error $e = (d - y)$
        Update weights $\vec{w} \to \vec{w} + \eta e \vec{x}_i$
        Update bias $b \to b + \eta(d - y)$
    **end for**
**end procedure**

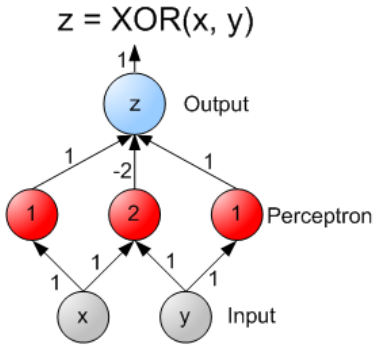



Figure 1.5: An example of a multilayer perceptron optimized for the XOR problem, where the data in $\mathbb{R}^2$ are labeled with the Boolean XOR value $x \oplus y = x + y \mod 2$ for $x, y = 0, 1$. [Pub. Dom.]

### *1.2.3 The Multilayer Perceptron*

The *multilayer perceptron* (MLP) can perform even more nontrivial classification and interpolation tasks than the SLP. As the name implies, the structure of the MLP includes more layers of neurons than the SLP.[3] Its structure is thus outlined as follows:

1. An input layer of $m$ signals, representing a data vector in $\mathbb{R}^m$, just like the SLP.

[3]For instance, the SLP cannot classify the "XOR problem" [2], since it is not linearly separable.

2. At least one *hidden layer*. A given hidden layer consists of a certain number (not necessarily the same across all hidden layers) of neurons, all of which take as input the output signals of the directly preceding layer. Each neuron $i$ is parametrized by its own vector $w_j$ of synaptic weights and its own bias $b$, and outputs its own signal $z$. Equivalently, the entire layer is parametrized by a *matrix* of weights $w_{ij}$ and a vector of biases $b_i$, and outputs a signal vector $z_i$. Thus, if $a$ is the vector of incoming signals, the preactivation output of the hidden layer is the vector

$$z_i = \sum_j w_{ij} a_j + b_i. \tag{1-20}$$

   The output of the layer is then the component-wise application of the activation function $\phi$, which can just be written as $\phi(z_i)$.

3. An output layer where the input signals are taken from the output of the final hidden layer, and the output signal $y$ is a vector. Similar to a hidden layer, it is parametrized by a weight matrix and bias vector.

Suppose the MLP has $L-1$ hidden layers. Then every parameter and output tensor in each layer in the MLP can be indexed by a superscript $(l)$, where $l$ is between $0$ and $L$, and where $l = 0$ means the input layer and $l = L$ means the output layer. Thus, for $l = 1, \ldots, L$ the preactivation outputs are written as

$$z_i^{(l)} = \sum_j w_{ij}^{(l)} a_j^{(l-1)} + b_i^{(l)}, \tag{1-21}$$

where $a^{(l-1)}$ is the output of the previous layer. In the base case $a^{(0)} = x$ is the input vector, and $a_i^{(l)} = \phi(z_i^{(l)})$ otherwise. In the output layer $y = a^{(L)}$.

### *1.2.4 Gradient Descent Training*

In the SLP, the error measure was simply the difference between the actual output $y$ and the desired output $d$. The error measure in the MLP can actually be a less simplistic measure, such as the norm $|y - d|^2$, which works with vector valued outputs. In general, we can use a differentiable *cost function* $\mathsf{c}(y, d)$ which is decreased as the output becomes closer to the desired output $d$. If $\theta$ is a parameter of the MLP, such as a weight or a bias, then the basis of *gradient descent* is to adjust the parameter in the direction that decreases the cost function, i.e.,

$$\theta \to \theta - \eta \frac{\partial \mathsf{c}}{\partial \theta}, \tag{1-22}$$

where $\eta$ is a training parameter called the *learning rate*. Calculating the partial derivative $\frac{\partial \mathsf{c}}{\partial \theta}$ in the MLP is not as nontrivial as it may seem, and can be done through the *backpropagation algorithm* given in Algorithm 2. The mathematical justification for the algorithm is given in the appendix.

With backpropagation, it becomes possible to efficiently implement *stochastic gradient descent* (Algorithm 3), which is simply a loop of backpropagation through a sufficient number of individual training examples until the cost function is no longer decreasing, which signals that learning has stopped.

A variety of modifications on the update rule have been developed to accelerate learning. The momentum method is a common improvement in which the parameter update $\Delta\theta(n)$ for the current step is corrected by its previous value $\Delta\theta(n-1)$ from the last update.

$$\Delta\theta(n) = \Delta\theta_{\text{backprop}}(n) + \alpha\Delta\theta(n-1) \tag{1-23}$$

where $\alpha$ is a decay factor between 0 and 1. The Adam method [24], which is the training algorithm used for this work, is yet another extension using exponentially decaying

**Algorithm 2** Backpropagation Algorithm for MLP Gradient

**procedure** MLP-BACKPROP(MLP weight matrices $w^{(l)}$ and bias vectors $b^{(l)}$ (for each layer $l = 1, \ldots L$), training input vector $x$, positive learning rate $\eta$ )
  Initialize vectors $z^{(1,\ldots,L)}$, $a^{(0,\ldots,L)}$, and $\delta^{(1,\ldots L)}$ appropriate to the sizes of the MLP layers.
  $a^{(0)} \to x$
  **for** $l = 1, 2, \ldots, L$ **do**
    $z_i^{(l)} \to \sum_j w_{ij}^{(l)} a_j^{(l-1)} + b_i^{(l)}$
    $a^{(l)} \to \phi(z^{(l)})$ ▷ Calculate the output of each layer (forward propagation)
  **end for**
  $\delta_i^{(L)} = \frac{\partial \mathsf{c}}{\partial a_i^{(L)}} \phi'(z_i^{(L)})$
  **for** $i = L-1, L-2, \ldots 1$ **do**
    $\Sigma \to \sum_j w_{ji}^{(l+1)} \delta_j^{(l+1)}$
    $\delta_i^{(l)} = \phi'(z_i^{(l)})\Sigma$
    $\Delta w_{ij}^{(l)} \to -\eta \delta_i^{(l)} x_j$
    $\Delta b_j^{(l)} \to -\eta \delta_i^{(l)}$
    $w_{ij}^{(l)} \to w_{ij}^{(l)} + \Delta w_{ij}^{(l)}$
    $b_i^{(l)} \to b_i^{(l)} + \Delta b_i^{(l)}$
  **end for**
**end procedure**

averages of the gradients and squared-gradients to adapt the update rules.

Another important modification to gradient descent is in the form of *mini-batch learning*, where each iteration evaluates the average of the cost function across multiple examples and averages the deltas, instead of just one example at a time. This smooths out the direction of the parameter gradient and its computation is easy to parallelize. The number of examples per iteration is called the *batch size*.

### *1.2.5 Convolutional Neural Networks*

A convolutional neural network (CNN) is a MLP that is specialized for pattern classification in images. The neurobiological motivation comes from a visual *receptive field*, i.e. a neuron does not receive all preceding signals as inputs, and restricts its attention

**Algorithm 3** Stochastic Gradient Descent Training Algorithm

1: **procedure** SGD(Initial weight matrices $w^{(l)}$ and bias vectors $b^{(l)}$, learning rate $\eta$, sequence of $n$ inputs $x_i$)
2: **for** $i = 1, 2, \ldots n$ **do**
3: MLP-Backprop($w, b, x_i, \eta$)
4: **end for**
5: **end procedure**

to a small cluster of them. Thus, instead of a layer being *fully connected* to its inputs, a more appropriate connectivity for a convolutional layer can be described, namely, by a convolution.

A 1D convolutional layer has preactivation values[4]

$$z_i^{(l)} = \sum_{j=0}^{k-1} w_j^{(l)} a_{i+j}^{(l-1)} + b_i^{(l)} \tag{1-24}$$

where $k$ is the length of the kernel vector $w$ which is indexed from $0$ for convenience.

In image classification, the input is naturally two-dimensional. Furthermore, it is desirable for a single layer to have the learning capacity for more than one feature. This can be done by increasing the rank of the output tensor to have a *channel index*. Channels naturally show up in images in the form of 3 (RGB) color channels. Thus, an image of size $N \times N$ is actually represented by a $N \times N \times 3$ input tensor.

For a 2D convolution, the input and output are furthermore expanded to have *spatial indices* $x$ and $y$. The preactivation output is thus given by

$$Z_{xyc} = \sum_{u,v=0}^{k} \sum_{d=0}^{C} W_{uvdc}\, A_{x+u,y+v,d} + B_{xyc}, \tag{1-25}$$

[4]In machine learning, the convolution follows an opposite sign convention that in other fields, using $a_{i+j}$ instead of $a_{i-j}$

where $C$ is the number of channels in the input tensor $A$, and $k$ is the spatial dimension length of the kernel $W$. The parameters of a 2D convolutional layer are thus the high rank tensors $W$ and $B$. The last two indices of $W$, the input channel and output channel indices, are specified in general when we would like to allow each output channel to depend on all the input channels of $A$.

Usually, a convolutional layer is followed by a *pooling step*, which groups small blocks of the convolution together in each channel and picks out a representative value, usually the maximum or average value. This step serves to reduce the size of the convolutional layer output.

A 3D convolutional neural network is yet a further natural generalization where the inputs form a 3D image. The parameters of a 3D convolutional layer are thus a rank 5 kernel tensor $W$ and a rank 4 bias tensor $B$. The neural network implemented for this work will be a 3D CNN.

### *1.2.6 Deep Neural Networks for Image Classification*

A general method of using a MLP/CNN as a classifier is to set the number of output neurons equal to the number of categories that one wishes to classify over. The category corresponding to the neuron with the largest response is then chosen as the inferred label for the input. For this task, one possible appropriate cost function is the $L^2$ norm

$$\mathsf{c}(y_i, d_i) = \sum_i |y_i - d_i|^2, \tag{1-26}$$

where $y_i$ is the output vector, and $d_i$ is the label vector whose entries are all $0$, except for the entry corresponding to the desired class, which is $1$. However, there is a more appropriate cost function with an information-theoretic basis.

Let $z_i$ be the preactivation signals of the output layer. Instead of applying an element-wise activation function on the $z_i$, we seek a transformation on $z_i$ whose output $y_i$ can be

interpreted as a *probability vector*, representing the "degrees of belief" of the neural network for each category. We could use a "hard" max function on the $z_i$, which transforms the largest element into $1$ and all others to $0$, but this is not continuous and throws away useful information about the neural network behavior. Instead, we should apply a "soft" max function that assigns a high probability to large signals and near-zero probabilities to lower signals. This is namely given by the *softmax* function

$$y_i = \frac{e^{z_i}}{\sum_j e^{z_j}}. \tag{1-27}$$

The softmax function could be seen as a Boltzmann distribution at a negative temperature, with the inverse interpretation that high "energy" states are favored. It also has the desirable property that if a number of outputs are very close together, then they have approximately the same likelihood.

While the $L^2$ cost function still quantifies the error between $y$ and $d$, the fact that both vectors represent probability distributions opens up a more natural cost function called the *cross-entropy* cost

$$\mathsf{c}(y_i, d_i) = -\frac{1}{n} \sum_i \left( d_i \ln y_i + (1 - d_i) \ln(1 - y_i) \right). \tag{1-28}$$

For classification tasks, the cross-entropy cost is not only a more natural cost function, it also speeds up learning faster than the $L^2$ cost.

Deep neural networks tend to train slowly or risk overfitting to the training data. In image classifying CNNs such as AlexNet, some possible optimizations that helped alleviate these issues include:

1. Using ReLU activation functions, especially over highly squashed functions like the sigmoid. (Such functions cause problematic vanishing gradients in backpropaga-

tion.)

2. Adding a penalty to the cost function proportional to the $L^2$ norm of the tensors of synaptic weights. For example, with a single fully connected weight matrix $w$, the new cost function is

$$\mathsf{c}' = \mathsf{c} + \lambda \sum_{ij} w_{ij}^2. \tag{1-29}$$

where $\lambda$ is a small constant called the *weight decay*. This technique is called $L^2$ *regularization*

3. Randomly picking about half of the neuron of a fully connected layer and zeroing out their output for each training step. This is called *dropout*.

4. "Xavier initialization" of the parameters [25]: When randomly initializing the weights to a layer, sample from a Gaussian distribution with variance $\sigma^2 = k/N_{\text{in}}$, where $N_{\text{in}}$ is the number of neurons in the previous layer, and $k = 1$ for the weights of a fully connected layer and $k = 2$ for the weights of a convolutional kernel.

5. "CReLU activation" for a convolutional kernel, which simply doubles the number of channels to add a ReLU activations that act on the negative preactivation signals $-z_{ic}$ [26].

### 1.3 Machine Learning on the Phases of the Ising Model

Carrasquilla and Melko in Ref. [3] implemented a MLP with a single hidden layer with 100 neurons for binary phase classification on the 2D Ising model. Using the array of spin configurations sampled from Monte Carlo simulations as inputs, the neural networks were trained to classify the snapshot as being in a ferromagnetic or paramagnetic state and thus identify a transition temperature $T_c$ where the classification probabilities cross. The neural network successfully discriminates between the FM and PM phases for system sizes

between $L = 10$ and $L = 60$, with natural classification error occurring in the vicinity of the phase transition temperature $T_c \approx 2.269$ . They find that the representation of the ferromagnetic phase in the MLP is naturally tied to observing the average magnetization $m$.

Additionally, they implement a CNN taught to identify the $T = 0$ and $T = \infty$ states of the Ising lattice gauge theory, which was analyzed to discuss the feasibility for CNNs to represent topological order. This study first showcased the potential for a CNN to perform nontrivial phase classification. A further study by Ch'ng *et. al.* [4] analyzes the generalization power of a 3D CNN phase classifier on the Fermi-Hubbard model for correlated electrons.

This work pursues a similar analysis for the 3D-EA spin glass model in zero and non-zero fields. The goal is to create a neural network that can accurately infer the phase of a spin glass model using input from Monte Carlo simulations. The next chapter will thoroughly introduce the Monte Carlo methods used in this study. Following that, the motivation, design, and results to date of the SG phase classifier implemented and tested here will be discussed.

# CHAPTER 2

# MONTE CARLO ALGORITHMS

Monte Carlo (MC) methods are some of the most important and common techniques in scientific computation, used in fields such as physics and chemistry. Some approximate methods can be hard to execute or inaccurate for very complex systems and problems in classical and quantum statistical physics. However, statistical physics is a problem domain that lends itself very easily to the Monte Carlo method, and thus amenable to thorough computational study. The following will outline the ideas and algorithms of Monte Carlo integration and Markov Chain Monte Carlo (MCMC), with emphasis on problems in statistical mechanics, and thus the simulation of spin glasses. The main goal is to introduce the Metropolis algorithm, first designed by Metropolis *et. al.* in 1953 [27], and the parallel tempering algorithm. The content of this section on Markov Chain Monte Carlo follows the introductions in Refs. [28–30].

## 2.1 Monte Carlo Integration

Performing accurate numerical integration becomes more difficult as the dimensionality of the domain increases. Simple-sampling Monte Carlo overcomes this "curse of dimensionality" by the independent random sampling of points, instead of using a systematic partitioning and quadrature as a deterministic numerical integration method would. An example procedure is outlined in Algorithm 4 when the domain of a function $f$ is the unit hypercube of the space. As the hypervolume of the domain is just 1, the integral of $f$ is simply its average value, which is easy to approximate through random samples of $f$ over its domain. [28, 31]

**Algorithm 4** Monte Carlo Integration

```
1: procedure MC-INT(int M, function f : (0,1)^d ⟶ ℝ)
2:     float I = 0
3:     for i = 1, 2 ... M do
4:         Generate a random vector x⃗ = rand(0,1)^d
5:         I += f(x⃗)
6:     end for
7:     return I/M.
8: end procedure
```

While numerical integration has deterministic error bounds that worsen with dimensionality, the convergence of simple Monte Carlo is based on the power of the Central Limit Theorem [32], and is thus independent of the domain dimension. This is stated in the following theorem:

**Theorem 1.** *Let $X$ be a uniformly-distributed random variable with values in the domain $V \subset \mathbb{R}^d$, which has a hypervolume $|V|$, and let $f : V \longrightarrow \mathbb{R}$ be a smooth function, and let the random variable $Y$ be defined by $Y = f(X)$. Then the integral $I$ of $f$ satisfies*

$$\langle Y \rangle = \frac{1}{|V|} \int_V \mathrm{d}x f(x) = I/|V|$$

*Furthermore, if $I$ is approximated from an independent sampling of $Y$ of size $M$, then the random error in the estimate is $\sigma_I^2 = |V|^2 \operatorname{Var} Y/M$.*

In practice, it should be very easy to sample vectors of uniform random numbers from the domain $V$. As a small demonstration of Monte Carlo integration, one can obtain an approximation for $\pi$ by random sampling. Here $f(\vec{x})$ is in the domain $(-1, 1) \times (-1, 1)$, $f(\vec{x}) = 1$ if $x^2 < 1$, and is $0$ otherwise. This specializes the MC integration algorithm into Algorithm 5, where we theoretically find $I = \pi r^2 = \pi$.

**Algorithm 5** Basic Monte Carlo $\pi$

```
1: procedure MC-PI(int M)
2:     int N_hits = 0
3:     for i = 1 ... M do
4:         int x = rand(−1, 1)
5:         int y = rand(−1, 1)
6:         if x^2 + y^2 < 1 then
7:             ++N_hits
8:         end if
9:     end for
10:    return N_hits/M.
11: end procedure
```

## 2.2 Markov Chain Monte Carlo

### *2.2.1 Statistical Mechanics and Monte Carlo*

In very high dimensional situations, integrating a function with a complex structure can be difficult. In statistical physics, we may be interested in calculating thermodynamic quantities of physical systems with a very large volume of possible configurations, which can be everyday objects such as a liter of ideal gas.[1] The set of all possible configurations of the large physical system is called the *state space* or *phase space* of the system. Each possible configuration, a completely determined arrangement of momentum, spin, energy, etc...[2] for each and every molecule is called a microstate of the gas.

When a physical system is in thermal equilibrium, to reiterate, the microstate $x$ of the system is a random variable following the *Boltzmann distribution*

$$P(x) = e^{-E(x)/T}/Z \tag{2-1}$$

[1] Which has $\sim 10^{23}$ molecules, each of which has a certain number degrees of freedom due to motion and internal molecular configurations

[2] For quantum systems, to the extent allowed by commuting observables.

where $E(x)$ is the energy of the microstate $x$, $T$ is the temperature of the system, and

$$Z = \int \mathrm{d}x e^{-E(x)/T} \tag{2-2}$$

is the partition function.[3] In the previous chapter, Eqs. (1-3) and (1-4) explicitly write out the Boltzmann constant $k$, although it is a mere proportionality factor between temperature and energy units. For convenience, it is taken to 1 from now on.

Often, instead of using the physical temperature, the Boltzmann distribution is written in terms of the inverse temperature $\beta = 1/T$ as a parameter.

$$P(x) = e^{-\beta E(x)}$$

A *macrostate* of the system is a thermodynamic quantity generally found by taking the expectation value of an observable $f(x)$ of the system with respect to the Boltzmann distribution.

$$\langle f \rangle = \int \mathrm{d}x \frac{e^{-\beta E(x)}}{Z} f(x)$$

If we were to attempt to evaluate $\langle f \rangle$ through simple MC, then the microstate $X$ would be a uniform random variable over the phase space, and our integrand would be the whole term above, not just $f(x)$. However, typically only a small subset of the phase space will make a significant contribution to calculating thermodynamic averages. The Boltzmann distribution damps out high energy states, and the number of low energy states is usually very small. Hence, it would be a waste to attempt to integrate over or even uniformly sample from the whole phase space. Furthermore, calculating $\langle f \rangle$ directly as above requires knowledge of $Z$, which may not be possible to compute analytically if we are resorting to

[3] An integral is normally equivalent to a sum. For a large enough number of microstates, integration is often more suitable.

numerical methods.

### *2.2.2 Markov Chains*

An *importance sampling* technique, where only a part of the phase space is preferentially sampled, can help alleviate Monte Carlo integration. *Markov Chain Monte Carlo* (MCMC) techniques perform importance sampling with a *guiding function* and a *Markov Chain* over the phase space. A Markov chain has the formal definition [29, 32]:

**Definition 1.** *A* Markov chain *is a sequence of random variable states* $(S_1, S_2, ...)$ *with a finite state space* $\Omega$ *that satisfies the* Markov property*: The probability distribution of a state* $S_{i+1}$ *can only be dependent on the previous state* $S_i$*, and on no other state in the sequence. That is,*

$$P(S_{i+1} = s_{i+1}|S_i = s_i, S_{i-1} = s_{i-1}, \ldots, S_1 = s_1) = P(S_{i+1} = s_{i+1}|S_i = s_i)$$

*for all steps* $i$*, with* $s_i \in \Omega$*. In addition, these conditional probabilities are the same at all steps. Thus, the behavior of a Markov Chain is entirely determined by a transition matrix* $p(s \to s')$ *such that*

$$p(s \to s') = P(S_{i+1} = s'|S_i = s)$$

*for all steps* $i$*. The powers of the transition matrix are written as* $p^n(s \to s')$*, which is the probability of reaching state* $s'$ *from* $s$ *in* $n$ *steps.*

Further definitions in Markov chain theory are give in the appendix. For reference, given any state $s$, the transition probabilities, being an entire set of conditional probabilities, will satisfy

$$\sum_{s'} p(s \to s') = 1. \tag{2-3}$$

It can be shown that an *ergodic* Markov chain, one that "mixes well" about its state

space, has an unique associated probability distribution $\pi(s)$ over its state space which satisfies

$$\pi(s) = \sum_{s'} p(s' \to s)\pi(s'). \tag{2-4}$$

$\pi(s)$ is called the *stationary distribution*, or *guiding function* in MCMC, of the Markov Chain, and it is the *long term* probability that the Markov Chain visits a particular state. In a MCMC simulation, one initiates a Markov chain over the phase space. For calculating a thermodynamic quantity, we require that $\pi(s)$ exists and be a Boltzmann distribution over the state space. If the Markov Chain has such a $\pi(s)$, then its state, *in the limit of infinite time between measurements*, is a random variable according to the Boltzmann distribution. Hence, if $\{f_i\}$ is a list of $N$ samples of $f(x)$ obtained through a Markov chain, then its mean

$$\langle f \rangle = \sum_i f_i/N \tag{2-5}$$

is an average over the Boltzmann distribution, i.e. a *thermodynamic average*.

Implementing the Markov chain efficiently requires and appropriate choice of transition probabilities. The Metropolis-Hastings algorithm is one popular implementation for Ising-like systems. It should be noted that since we can't actually sample between infinite times, measurements from a Markov chain will suffer from *correlation*, and this will call for a more involved error analysis than for simple MC. [28, 29, 31]

### *2.2.3 The Metropolis-Hastings Algorithm*

A MCMC simulation can be implemented efficiently by splitting up state transitions into two stages: *proposal*, and *acceptance*. If the Markov chain is in a state $s$, then the probability that the algorithm *proposes* to move to a new state $s'$ is fully specified by a transition matrix $g(s \to s')$. However, the probability that the Markov chain actually *moves* to the new state $s'$ is specified by another transition matrix $A(s \to s')$, called the

acceptance probabilities. The move can otherwise be rejected, and so the Markov chain stays in $s$. In all, the transition probabilities are given by the product

$$p(s \to s') = g(s \to s')A(s \to s') \tag{2-6}$$

**Algorithm 6** Markov Chain Monte Carlo

**procedure** MCMC(Phase space/Markov chain states $S$, energy function $E(s)$, proposal and acceptance matrices $g$ and $A$)
    Initialize the Markov chain at some state $s$
    **loop**
        Propose a new state $s'$ with probability $g(s \to s')$
        Generate a random number $x$ between 0 and 1
        **if** $x < A(s \to s')$ **then**
            Accept move $s \to s'$
        **else**
            Reject move and stay in $s$.
        **end if**
        Measure observable $f(s)$.
    **end loop**
    Return mean of observed values of $f$.
**end procedure**

As shown in the appendix, the Metropolis-Hastings algorithm makes a choice of transition probabilities for a reversible Markov Chain that turns out to be very general and powerful. If $\pi(x)$ is the desired stationary distribution, then we choose

$$A(s \to s') = \min\left(1, \frac{\pi(s')g(s' \to s)}{\pi(s)g(s \to s')}\right). \tag{2-7}$$

The Metropolis algorithm is especially simple and efficient in statistical mechanics because the ratios of the Boltzmann distribution distribution will not involve the parti-

tion function. Assuming a symmetric proposal probability matrix, which the Metropolis algorithm will satisfy, the acceptance probabilities thus specialize to

$$A(s \to s') = \min\left(1, e^{-\beta \Delta E}\right). \tag{2-8}$$

where $\Delta E = H(s') - H(s)$ is the energy change associated with the proposed move.

It should be mentioned that other choices of acceptance probabilities of the form (2-7) are possible, resulting in algorithms other than the Metropolis algorithm, one example being the heat bath algorithm [28].

### *2.2.4 Example: Metropolis Algorithm for the Ising Model*

Algorithm 7 tailors the Metropolis algorithm for the Ising model, and shows the energy and net magnetization $E$ and $M$ as observables of interest. Adaptation for other observables is straightforward. In Markov Chain Monte Carlo for Ising-like models, the unit of time usually chosen to represent the progress of the algorithm is the *sweep*, which is equal to $N$ attempts of a Metropolis update, where $N$ is the system size.

## 2.3 Replica Exchange Monte Carlo (Parallel Tempering)

### *2.3.1 Replica Exchange*

It is straightforward to adapt the Metropolis algorithm for simulating a spin glass instance by adjusting the calculations for $\Delta E$. As previously discussed, spin glasses are characterized by a complex energy landscape with metastable states. As a result of these metastable states, the low-temperature spin glass phase see significant energy barriers between "energy valleys" of microstates. If one were to try to run a spin glass Markov chain at a single low temperature, it is highly likely to get stuck in one of these energy valleys and almost never visit another one. This is a condition of *broken ergodicity* in the statistical mechanics of the system, and thus of the Markov chain.

**Algorithm 7** Metropolis algorithm for the 2D Ising Model

**procedure** ISING-METROPOLIS-2D(Spin array $s$ with length $N$, temperature $\beta$)
    Initialize spin array $s$ with random values $s_i = \pm 1$
    Initialize energy $E = -J \sum_{\langle i\, j \rangle} s_i s_j$
    Initialize magnetization $m = \sum_i s_i$
    **loop**
        **loop**(N times = One "sweep")
            Pick a random spin index $i$ between $0$ and $N - 1$.
            Calculate energy change $\Delta E$ associated with flipping the spin $s_i \to -s_i$.
            Generate a random number $x$ between $0$ and $1$
            **if** $\Delta E < 0$ or $x < e^{-\beta \Delta E}$ **then**
                Flip spin $s_i \to -s_i$
                Update $E \to E + \Delta E$ and $M \to M + 2s_i$
            **else**
                Reject move and stay in $s$.
            **end if**
        **end loop**
        Measure observable $E$ and $M$.
    **end loop**
    Return mean of observed values of $f$.
**end procedure**

Ergodicity breaking in spin glass models makes MCMC dramatically more difficult and slow. High temperatures Markov chains, on the other hand, will mix well. One possible solution is given by the parallel tempering algorithm [33], which permits the low-temperature Markov chains to escape energy valleys. Consider the simple case where a single Hamiltonian $H$ is being simulated simultaneously at different temperatures $\beta_1$ and $\beta_2$ in two independent Markov chains. The joint stationary probability distribution of the two simulation configurations is simply a product of the two Boltzmann distribution at each temperature. Hence, the normalization factor $Z$ is the product of the partition functions for the two temperatures $Z_1$ and $Z_2$, and the probability that the simulation has the

respective configurations $\vec{s}_1$ and $\vec{s}_2$ is proportional to

$$P(\vec{s}_1, \vec{s}_2) \propto e^{-\beta_1 H(\vec{s}_1) - \beta_2 H(\vec{s}_2)} \tag{2-9}$$

Consider a Monte-Carlo-like move where the configurations $\vec{s}_1$ and $\vec{s}_2$ are *swapped* between the two Markov chains. Then the proofs in the appendix leading to detailed balance can be modified by using $\pi(s) = P(\vec{s}_1, \vec{s}_2)$ to show that the acceptance probability

$$A_{\text{swap}} = \min[1, e^{(\beta_1 - \beta_2)(E_1 - E_2)}] \tag{2-10}$$

allows such a move to satisfy detailed balance. This is called a *replica exchange move*, and forms the basis of the *parallel tempering algorithm* (PT) described in Algorithm 8.

Instead of performing independent MCMC simulations at each desired temperature, all the Markov chains are simulated simultaneously. After all replicas have undergone one sweep of the Metropolis algorithm, the replica exchange move is attempted for every neighboring temperature. (We will call a set of replicas assigned to each temperature subject to exchanges with each other a *parallel tempering chain*.) Therefore, the parallel tempering algorithm has the capacity to bring replicas of a spin glass instance in and out of energy valleys and restore ergodicity for the low temperature Markov chains.

Recall that the overlap parameter $q$ is defined in Eq. (1-12) in terms of two replicas of the Hamiltonian instance at the same temperature. Thus, we still need two independent Markov chains to calculate $q$. This calls for another entirely independent PT chain to ensure the measurements are not correlated.[4] Algorithm 8 takes into account the possibility of simulating multiple ($K$) independent chains.

While improved variations of PT exist, only replica exchange is the most efficient

[4]Whereas one PT chain would suffice for the Ising model.

known way for simulating spin glasses in a non-zero field due to chaotic dynamics [11].

**Algorithm 8** Parallel Tempering Algorithm

**procedure** PARALLEL-TEMPERING(Inverse temperature array $\beta$ with length $M$. $M \times K \times N$ spin tensor $S$.)
    **loop**
        **for** $i$ in $0 : M$ **do**
            **for** $j$ in $0 : K$ **do**
                Perform one Metropolis sweep on spin array $S_{ij}$ at $\beta_i$.
            **end for**
        **end for**
        **for** $i$ in $0 : M - 1$ **do**
            **for** $j$ in $0 : K$ **do**
                $\Delta \to (\beta_i - \beta_{i+1})(E(S[i, j]) - E(S[i + 1, j]))$
                Generate a random number $x$ between 0 and 1
                **if** $\Delta > 1$ or $x < e^{\Delta}$ **then**
                    Swap arrays
                **end if**
            **end for**
        **end for**
        Measure observables, e.g. the overlap $q_k = \sum_i S_{k0i} S_{k1i}$ at each temperature $\beta_k$.
    **end loop**
**end procedure**

# CHAPTER 3

# TERNARY CNN PHASE CLASSIFIER FOR 3D SPIN GLASS MODELS

## 3.1 Design Motivations

1. As previously stated, the spin configurations snapshots of the 3D-EA model at low temperatures are visually indistinguishable from paramagnetic states. To overcome this, the inputs to the neural network are *replica overlap configurations* (ROCs) which are the 3D images of the *overlap units*

$$q_i = s_i^{(1)} s_i^{(2)} \tag{3-1}$$

taken from two independent Markov chains. The usual overlap parameter is the sum of the $q_i$. The overlap units for large $T$ will be similar to the spin configuration in that each $q_i$ will be uncorrelated from the rest.

2. The networks in Ref. [3] infer a phase using a single Monte Carlo configuration sample at a time as the input to the neural network. It is proposed here that reliable phase classification should be a function of *multiple configuration samples* from a Monte Carlo simulation taken together as one example. The neural network then performs an averaging step (*thermal averaging*) over the samples.

3. For the neural network to learn a reasonable representation for the spin glass phase, it needs to consider configurations across *multiple instances* simulated at the same temperature $T$ and with the same field strength $h$, as the critical properties for a spin glass transition are inherently due to such instance averages. Thus, it is proposed here that an averaging step over a sample of instances (*instance averaging*) is

necessary for a neural network to possibly classify the spin glass phase faithfully.

4. It is possible that a binary classifier between the SG and PM phases may learn to represent the SG state in a way that is only sensitive to the magnitude of the overlap parameter $q$. While such a classifier would probably be successful in a zero-field model, one could question whether such a classifier has truly "learned" the spin glass state. However, teaching a classifier to distinguish between the SG and FM phases could help create a better knowledge representation of the SG state that is not too directly tied to $|q|$. This is because, for both a typical spin glass replica and for the Ising ferromagnet, there is a large probability of $q \approx \pm 1$ below the critical point. However, intermediate value of $q$ between $-1$ and $1$ are likely only in a spin glass model due to nontrivial ergodicity breaking. Thus, it is proposed here that a spin glass phase classifier should be taught *three categories*, i.e. a ternary classifier, of states (PM, FM, and SG) in 3D spin models.[1]

5. When generalizing to the non-zero field case, the means of each $q_i$ will be biased towards $1$ as the spins become more likely to point in the direction of their local field. Thus, each $q_i$ should be centered around its thermal mean $\langle q_i \rangle$. Here, each overlap unit is also normalized by its variance. Aside from adding a negligible amount of noise, this procedure does not affect the zero field inputs due to spin-flip symmetry. It will, however, significantly transform the non-zero field inputs.

### 3.2 Architecture Outline

The Classifier is a deep 3D convolutional neural network with 5 convolutional layers, 2 intermediate averaging steps, and a thermal normalization layer. Its output is a softmax probability distribution of 3 classes, classifying among the PM, FM, or SG phases.

[1] However, due to point 1, distinguishing between, for example, FM and PM phases will not be possible, since the ROCs for both states are the same. [15]

Algorithm 9 describeds the procedure for the neural network.

*Implementation Note*: For simulating convolutions with periodic boundary conditions, the configurations are manually padded and all convolutions assume "valid" boundary conditions (the kernel only convolves where it is never out of bounds). No pooling steps are used. (The utility of non-pooling CNNs is analyzed in Ref. [34]. Also, the periodic boundary conditions of the models allow for very liberal control of the convolution output size without the need for pooling.) Where not specified, the kernel stride is 1 in all directions and the kernel length is 2.

## 3.3 Implementation and Training Procedure

The neural network was implemented using the *Tensorflow* library [35] in the Python programming language. Tensorflow works through the specification of the computation graph of the neural network, which is then followed by looping through the input data to feed into the computation graph. A training loop can be constructed to modify the parameters of the neural network in each layer through a gradient descent learning algorithm.

### *3.3.1 Training Input*

The computation graph inputs are drawn from the output data of $K = 2$ PT simulations of the 3D-EA model and the 3D Ising model. The system sizes for training and most evaluations is $L = 8$. The PT simulations are parametrized as follows:

1. 3D-EA: $N = 250$ instances simulated for $2^{22}$ sweeps over $M = 16$ temperatures. From each instance, $2R = 64$ *spin* configurations were sampled after 50% progress (for thermalization). (32 spin configurations were sampled per temperature, per PT chain, per instance.)

2. 3D Ising: Simulated for $2^{20}$ sweeps over $M = 32$ temperatures. After 50% progress, spin configurations were sampled every $2^{10}$ sweeps. A total of $512$ spin configurations

**Algorithm 9** Deep 3D convolutional neural network for ternary phase classification on an Ising-like 3D periodic square lattice.

**procedure** SG-NET-EVAL(Constants: Batch size $B$, $Q$ instances per example, $R$ overlap configuration samples per instance; Input: $B \times Q \times R \times L^3$ tensor $T$ of overlap units )
    **Thermal Normalization**: Standardize elements of $T$ with mean and s.d across the sample dimension $R$.
    **if** $L \leq 8$ **then**
        **Periodic-pad** $T$ configurations to size $9^3$
    **end if**
    **if** $L \leq 12$ **then**
        **Periodic-pad** $T$ configurations to size $13^3$
    **end if**
    *Comment*: $T$ dimensions $B \times Q \times R \times (9^3 \text{ or } 13^3) \times 1$
    **Convolution 1A**: Kernel size $3^3$; Convolution stride of 2; 32 output channels with CReLU activation (effectively 64 output channels); $T$ dimensions $B \times Q \times R \times (4^3 \text{ or } 6^3) \times 64$
    **Periodic-pad** configurations by 2 in all spatial dimensions. $T$ dimensions $B \times Q \times R \times (6^3 \text{ or } 8^3) \times 64$
    **Convolution 1B**: 64 output channels with CReLU activation (effectively 128 output channels); $T$ dimensions $B \times Q \times R \times (5^3 \text{ or } 7^3) \times 128$
    **Average** across sample dimension $R$. $T$ dimensions $B \times Q \times (5^3 \text{ or } 7^3) \times 128$
    **Convolution 2A**: 64 output channels with ReLU activation; $T$ dimensions $B \times Q \times (4^3 \text{ or } 6^3) \times 64$
    **Convolution 2B**: Convolution stride of 2; 32 output channels with ReLU activation; $T$ dimensions $B \times Q \times (2^3 \text{ or } 3^3) \times 32$
    **Average** across instance dimension $Q$. $T$ dimensions $B \times (2^3 \text{ or } 3^3) \times 32$
    **Periodic-pad** configurations by 1 if lengths are odd. $T$ dimensions $B \times (2^3 \text{ or } 4^3) \times 32$
    **Convolution 3**: 256 output channels with ReLU activation; $T$ dimensions $B \times (1^3 \text{ or } 3^3) \times 256$
    **Average** across spatial dimensions of configurations. $T$ dimensions $B \times 256$
    **Output Layer**: $256 \to 3$. No activation. $T$ dimensions $B \times 3$
    **Softmax** on the elements of $T$.
    **return** $T$
**end procedure**

were sampled per temperature, per PT chain.

During data pre-processing in the computation graph, the spin configurations are shuffled (with those from the same PT chain) and then paired up to produce the ROC groups as input to the Classifier. Each input is labeled with a class number $0, 1$ or $2$ corresponding to the PM, FM, or SG phase respectively.

The class labels assume an exactly known value of $T_c$. While the transition temperature is known to high precision for the 3D ferromagnet, some error and variation exists for calculated values of the transition temperature for the 3D-EA. The value of $T_c = 0.951$ found by Katzgraber *et. al.* [5] has been assumed for determining the SG phase label.

### *3.3.2 Training Procedure*

Training the Classifier used 5000 steps of the Adam method [24] with a learning rate of $10^{-5}$ and recommended defaults for all other parameters of the method. During training, a batch consisted of $B = 16$ examples, and a single input example to the Classifier consisted of $Q = 8$ instances with $R = 16$ ROCs per instance. Every example has a 50% chance of being a ferromagnet sample and a 50% chance of being a 3D-EA input. Then the instance (if 3D-EA) and temperature indices are randomly determined for the example. [2] Ferromagnetic samples are also grouped as though they were drawn from $Q = 8$ instances. Though this grouping is a pointless one, it keeps the input to the Classifier consistent.

The cost function includes $L_2$ regularization terms with a weight decay of $0.001$ on all convolutional kernels and the fully connected weights to the output layer. The weights were initialized using Xavier initialization, as described in Sec. 1.2.6.

[2] Thus, the examples represent a roughly 2:1:1 ratio of the phases.

# CHAPTER 4

# RESULTS AND DISCUSSION

## 4.1 Zero Field Spin Glasses Evaluation

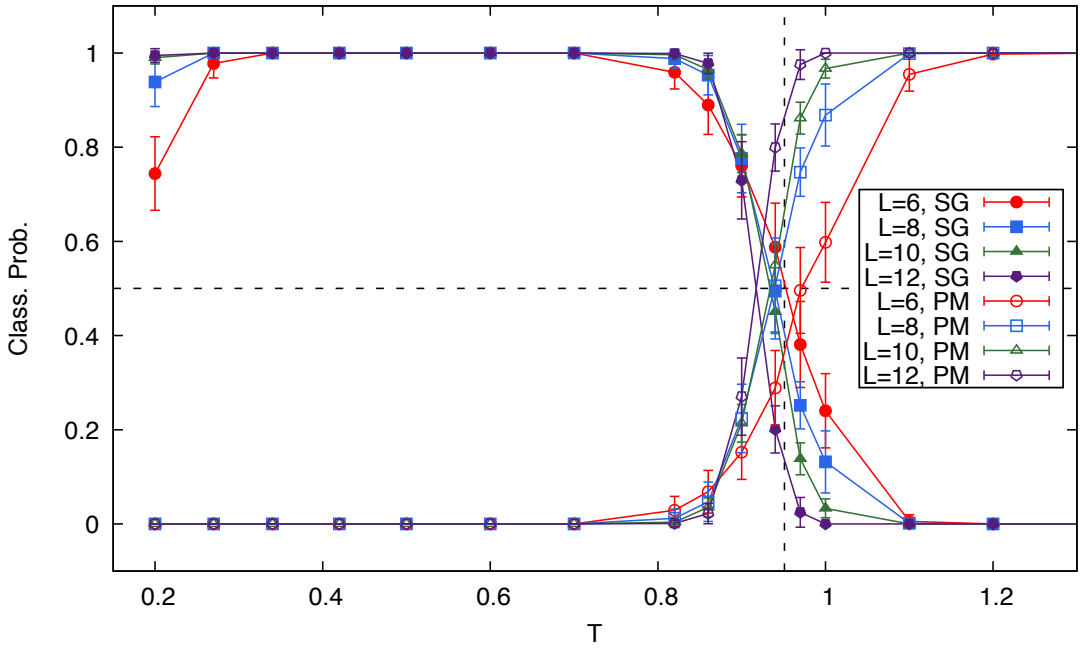
### *4.1.1 3D-EA and 3D Ising Ferromagnet*

Figure 4.1 shows the classification probabilities of the Classifier evaluated on a sampling of spin glass test instances from Monte Carlo simulations, independent of the training data.[1] ($N = 250$ for $L = 8$, and $N = 50$ otherwise.) Throughout the figures, vertical dashed lines represent transition temperatures and horizontal dashed lines the 50% probability line. The classification probabilities follow a smooth transition near the critical temperature. The Classifier performs well except for an anomalous jump in FM classification at the lowest temperature. This is likely due to the possibility of sampling instances with simple energy landscapes when dealing with finite-size spin glasses. This is supported by observing the finite size behavior as $L$ increases to $12$, for which the FM anomaly vanishes.

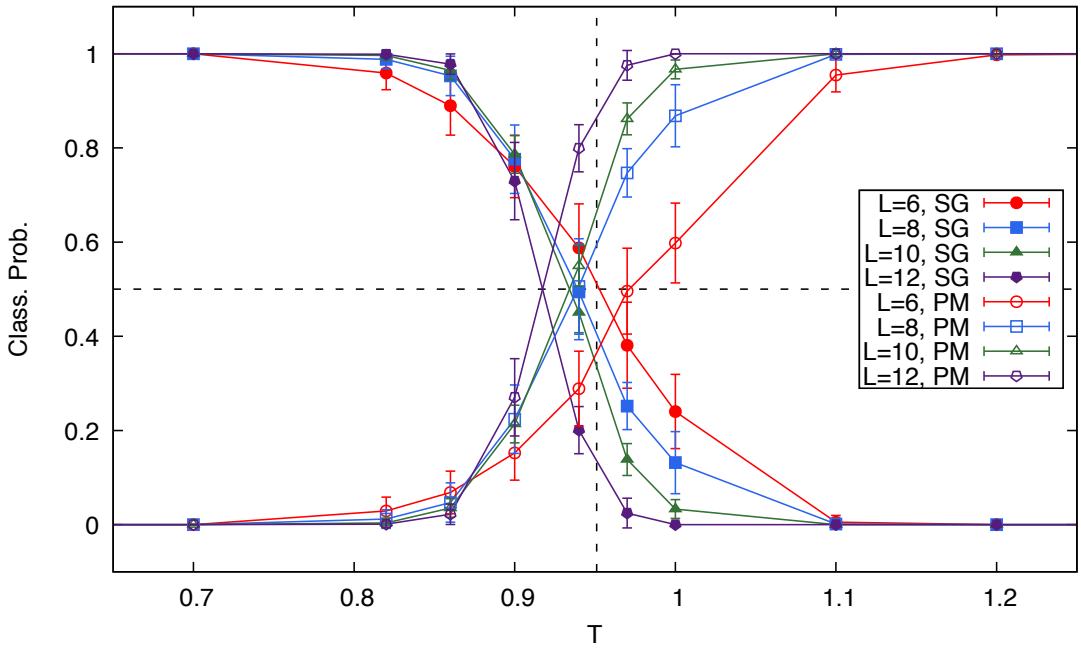
### *4.1.2 Bimodal Interaction 3D-EA Model*

As a secondary generalization test, the Classifier is also evaluated on samples of $N = 50$ instances of the zero field 3D-EA with bimodal disorder (as opposed to Gaussian disorder). It appears to closely identify the bimodal transition temperature calculated in the same work by Katzgraber *et. al.* that found the Gaussian transition temperature that was assumed for the SG classification labels during training. We take this result as good evidence that the Classifier has a distinctive and consistent internal representation of the spin glass phase.

[1] All testing data for this chapter was simulated with $M = 20$ temperature PT to better observe the classification curves near $T_c$.




(a) Zoom in up to $T = 1.2$.




(b) Zoom in between $T = 0.7$ and $1.2$

Figure 4.1: Classification probabilities of the neural network on 3D-EA test samples. The dashed vertical line is at the assumed transition temperature $T_{c,SG} = 0.951$ [5]. FM curves are not shown for space, but are implied nonzero near $T = 0.2$.

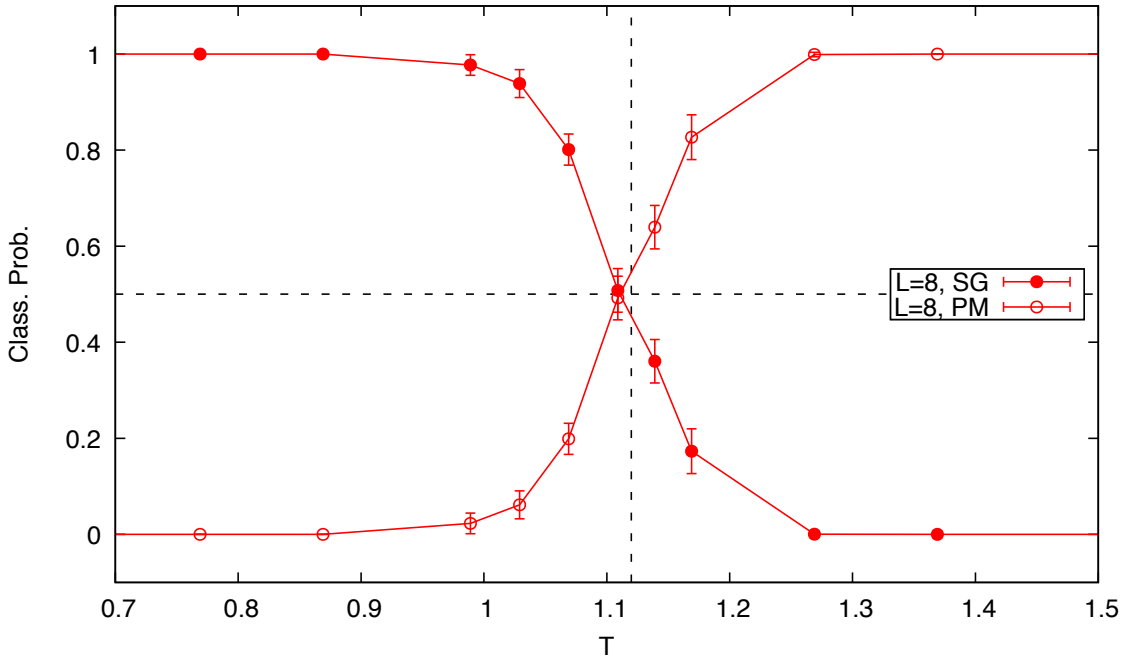



Figure 4.2: Classification probabilities of the neural network evaluated on the 3D-EA model with bimodal-distributed ($\pm 1$) interactions. The neural network successfully identifies a transition near the temperature $T_{c,BM} = 1.120$ also found in Ref. [5].

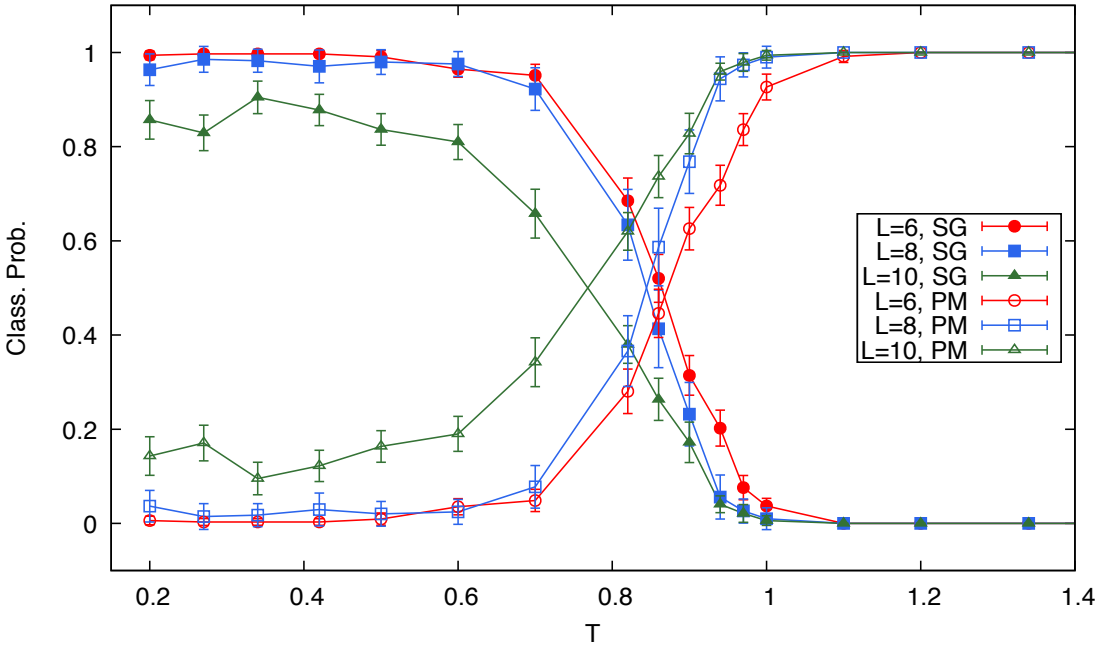
## 4.2 Non-zero Field Spin Glass

The classification probabilities for the non-zero field 3D-EA model are shown in figures 4.3 and 4.4. ($N = 50$ instances for each curve.) It can be observed that even at $H = 0.05$, the spin glass signal for the Classifier begins to collapse, although the classification probability is still large. At strong fields $H \geq 0.15$, the paramagnetic phase becomes dominant. It is worth analyzing the finite-size behavior of an intermediate field value to help elucidate whether this behavior is a finite-size artifact, or if it is part of a true transition. The classification probabilities for $H = 0.10$ shown in Figure 4.3b illustrate a significant drift of the crossing temperature between $L = 6$ and $L = 10$, along with evidence that it drifts further to $T = 0$ and becomes strongly confused between PM and

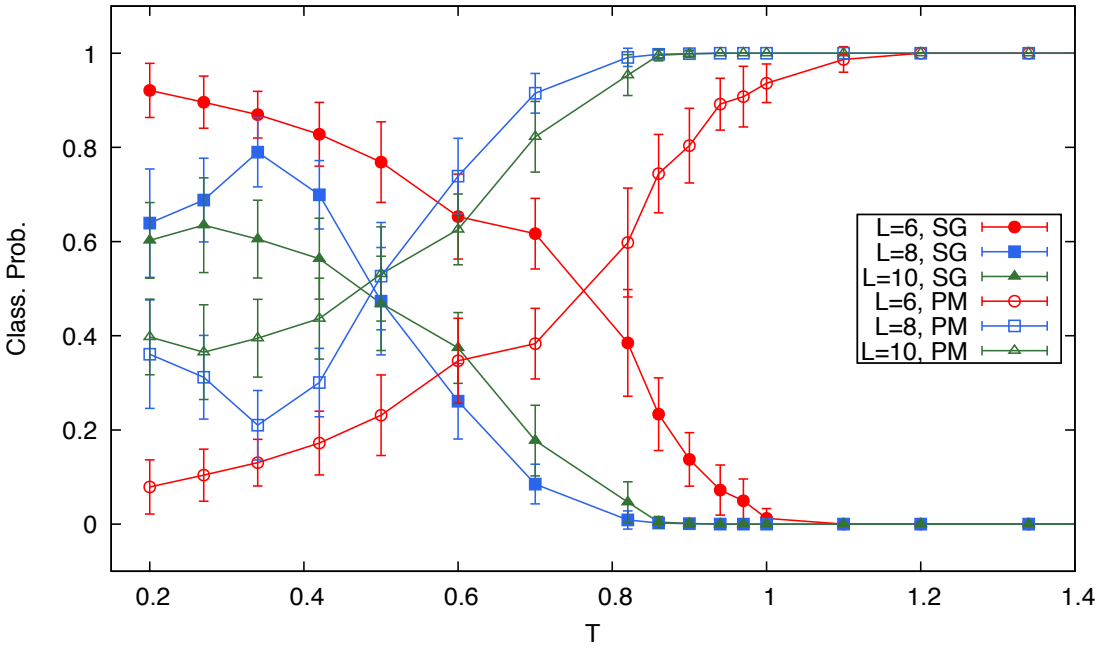
SG classification as the system size increases.

Figure 4.5 shows the progression of the SG classification curves as the field strength is increased for each system size $L = 6, 8$ and $10$. ($N = 25$ for $H = 0.30$, $N = 50$ otherwise.) While the zero field curve keeps a distinct transition region, all of the non-zero field curves show evidence of dying away in the thermodynamic limit down to $H = 0.05$.

Overall, with the data available thus far up to the system sizes studied, it is not evident that the spin glass phase persists at non-zero fields in the thermodynamic limit, according to the Classifier's inferences.




(a) At $H = 0.05$.




(b) At $H = 0.10$.

Figure 4.3: Classification probabilities at low and intermediate fields.

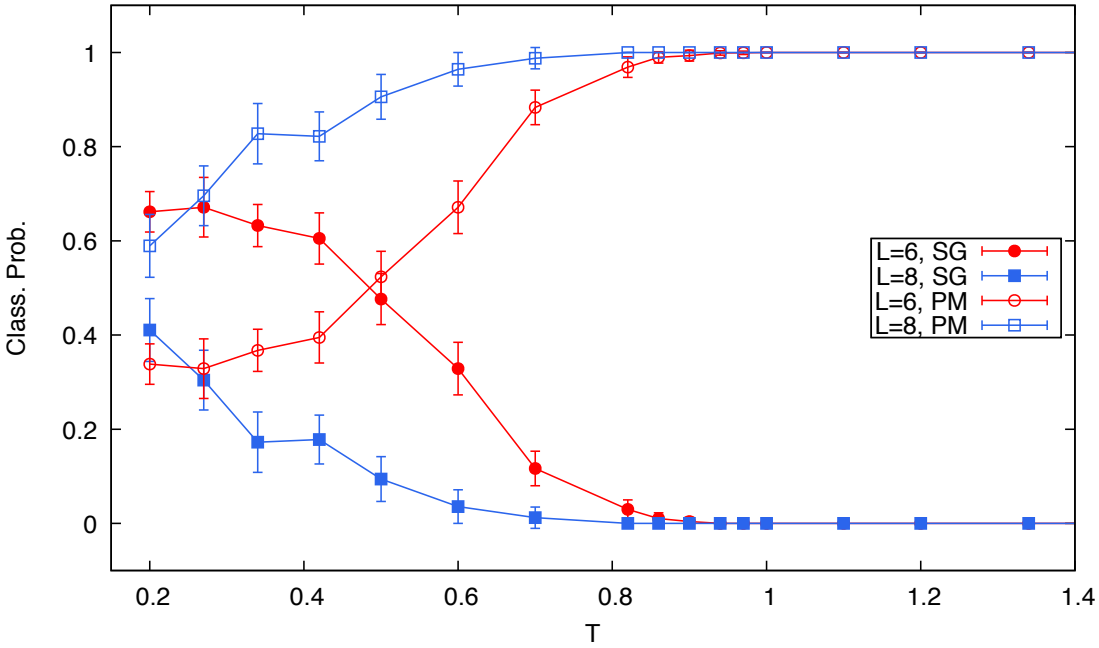
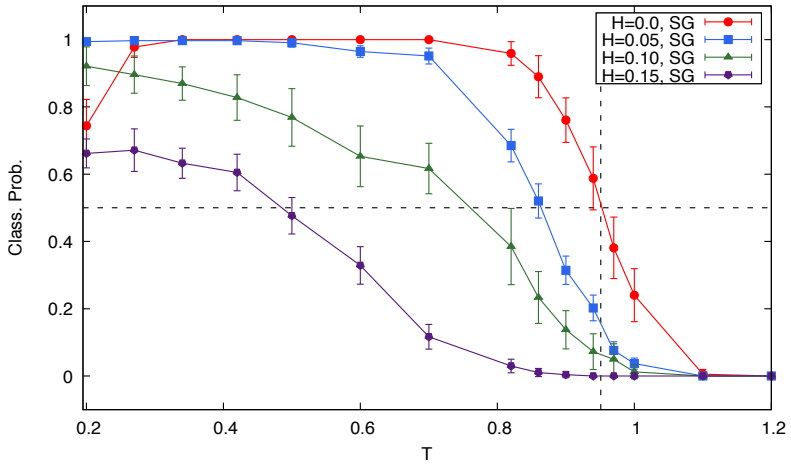
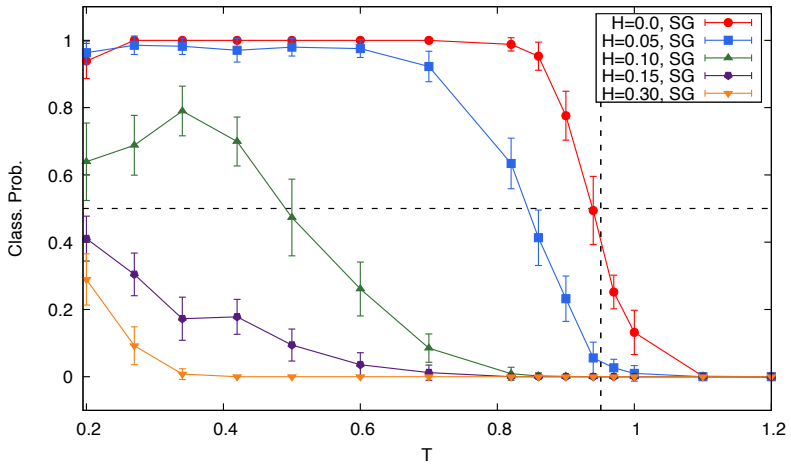


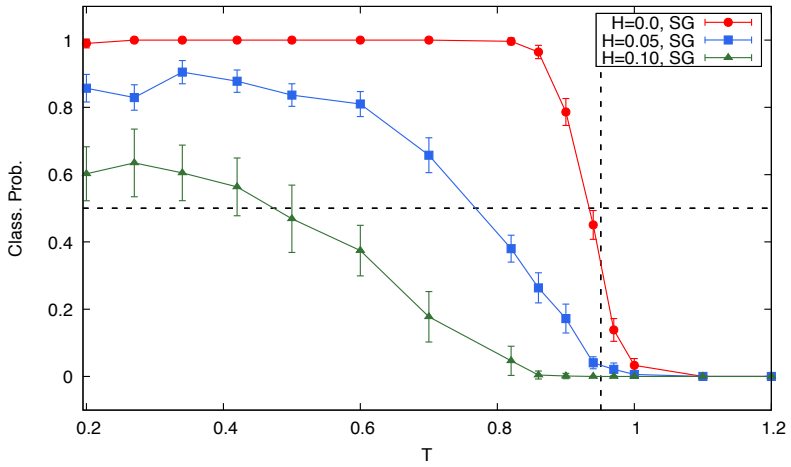

Figure 4.4: Classification probabilities at strong fields. ($H = 0.15$)




(a) Progression through fields for $L = 6$.




(b) Progression through fields up to $h = 0.30$ for $L = 8$.




(c) Progression through fields for $L = 10$.

Figure 4.5

### 4.3 Limitations of the Classifier and its Evaluation

One possible criticism of the Classifier is that it may be too "deep" for the purpose of distinguishing among three categories. Indeed, the gradient statistics showed that the weight histograms of the kernels in the first two convolutions changed very little with a combination of small gradients and a small learning rate. This may be a sign that further fine-tuning of the training parameters was required, or perhaps that at least one convolutional layer can be removed without appreciably affecting performance.

The error bars in the testing results of this chapter represent the statistical error of evaluating the Classifier on a fixed set of samples from (mostly) $N = 50$ instances at each field strength by a random sampling procedure of temperature index, instances, and spin configuration pairs to produce the input examples to the Classifier (similar to sampling during training). Further analysis on the error for $N > 50$ instances may be desirable, if not to make sure that the number of instances is sufficient, then to perhaps point out the relatively low number of instances needed as an advantage of neural network methods. Another form of further helpful error analysis may be repeating the training process multiple times and averaging the output of the classifier. To ensure decorrelation, independent spin glass instances could be utilized for the training data for each Classifier.

While the Classifier has a clear potential for accurate identification of the spin glass phase in zero field models, some caution should be exercised in interpreting the predictions at nonzero fields. Ideally, the high SG probabilities up to $H = 0.05$ could be attributed to a finite-size transition, but there is no concrete evidence at the moment that the Classifier is not making a generalization error. Furthermore, in Ref. [11], $H = 0.05$ was the lowest field studied down to which there should evidently be no transition. We would ideally like the classifier to identify $H = 0.05$ as paramagnetic, or at least SG/PM confused.

It should be emphasized that the Classifier architecture is committed to a three-dimensional

representation, and hence is not expected to give meaningful output on 2D or mean-field models.

## 4.4 Concluding Remarks

This work implemented and evaluated a 3D convolutional neural network for ternary phase classification in Ising-like models on a 3D square lattice. Current data shows promising accuracy for the 3D-EA models, as well as plausible generalization power in the spin glass model with the addition of random fields. While machine learning methods may not be stand-in replacements for Monte Carlo methods, nor are they as theoretically robust, they could open up new directions to current questions in the study of spin glasses. For example, another application currently being investigated is reentrant phenomena in the 3D Ising model [36]. In general, these methods once optimized could be used to probe the phase characteristics of modified spin glass models where Monte Carlo statistics may happen to be unavailable or inconclusive. This work focused on an application where such statistics were previously studied, and a situation where MC simulation of spin glasses alone has not been sufficient is yet to be encountered by or known to the author. However, one possibility where neural network methods on lattice systems could prove of immense practicality and benefit is control components in Monte Carlo algorithms. One good starting point could be ML-based feedback-optimization for parallel tempering, which at the moment only has some iterative optimizations [37].

Future work based on the results presented here will focus on fine-tuning the architecture and training process, improving statistics with more instance samples, and ensuring that the results of training are reproducible.

Further computational and theoretical research in this disciplinary interface may be encouraging. Both neural networks and spin glasses are examples of complex systems with a breadth of applications, where in fact some neural networks can find a description

though long-range spin glasses [15]. The converse study of describing spin glasses through neural networks may prove rewarding from the point of view of complexity theory.

## REFERENCES


[1] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning", Book in preparation for MIT Press, 2016.

[2] S. O. Haykin, *Neural networks and learning machines* (Pearson, 2008).

[3] J. Carrasquilla and R. G. Melko, "Machine learning phases of matter", Nat. Phys. AOP (2017).

[4] K. Ch'ng, J. Carrasquilla, R. G. Melko, and E. Khatami, "Machine learning phases of strongly correlated fermions", arXiv:1609.02552, 2016.

[5] H. G. Katzgraber, M. Körner, and A. P. Young, "Universality in three-dimensional ising spin glasses: a monte carlo study", Phys. Rev. B **73** (2006).

[6] A. N. W and N. D. Mermin, *Solid state physics* (Cengage, 1976).

[7] K. Binder and A. P. Young, "Spin Glasses: Experimental Facts, Theoretical Concepts and Open Questions", Rev. Mod. Phys. **58**, 801 (1986).

[8] E. Ising, "Beitrag zur Theorie des Ferromagnetismus", Z. Phys. **31**, 253 (1925).

[9] L. Onsager, "Crystal Statistics. I. A Two-Dimensional Model with an Order-Disorder Transition", Phys. Rev. **65**, 117 (1944).

[10] S. F. Edwards and P. W. Anderson, "Theory of spin glasses", J. Phys. F: Met. Phys. **5**, 965 (1975).

[11] A. P. Young and H. G. Katzgraber, "Absence of an Almeida-Thouless line in three-dimensional spin glasses", Phys. Rev. Lett. **93** (2004).

[12] L. D. Landau and E. M. Lifshitz, *Statistical physics, part 1*, 3rd, Vol. 5, Course of Theoretical Physics (Butterworth-Heinemann, 1980).

[13] J. M. Yeomans, *Statistical Mechanics of Phase Transitions* (Oxford University Press, Oxford, 1992).

[14] K. Binder, "Critical properties from Monte Carlo coarse graining and renormalization", Phys. Rev. Lett. **47**, 693 (1981).

[15] H. Nishimori, *Statistical physics of spin glasses and information processing: an introduction* (Oxford University Press, 2001).

[16] K. H. Fischer and J. A. Hertz, *Spin glasses* (Cambridge University Press, 1991).

[17] D. S. Fisher and D. A. Huse, "Absence of many states in realistic spin glasses", J. Phys. A **20**, L1005 (1987).

[18] G. Parisi, "Infinite number of order parameters for spin-glasses", Phys. Rev. Lett. **43**, 1754 (1979).

[19] D. Sherrington and S. Kirkpatrick, "Solvable model of a spin glass", Phys. Rev. Lett. **35**, 1792 (1975).

[20] T. Jörg, H. G. Katzgraber, and F. Krzakala, "Behavior of Ising Spin Glasses in a Magnetic Field", Phys. Rev. Lett. **100**, 197202 (2008).

[21] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity", Bulletin of Mathematical Biophysics **5**, 115–133 (1943).

[22] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain", Psychological Review **65**, 386–408 (1958).

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", in Neural information processing systems (2012).

[24] D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization", arXiv:1412.6980v9.

[25] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks", in Jmlr workshop and conference proceedings, Vol. 9 (2010).

[26] W. Shang, K. Sohn, D. Almeida, and H. Lee, "Understanding and improving convolutional neural networks via concatenated rectified linear units", arXiv:1603.05201v2.

[27] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines", J. Chem. Phys. **21** (1953).

[28] H. G. Katzgraber, "Introduction to monte carlo methods", arXiv:0905.1629v3.

[29] M. E. J. Newman and G. T. Barkema, *Monte carlo methods in statistical physics* (Oxford University Press, 1999).

[30] J. M. Thijssen, *Computational physics* (Cambridge University Press, 2017).

[31] W. Krauth, *Statistical mechanics: algorithms and computations* (Oxford University Press, 2006).

[32] D. P. Bertsekas and J. N. Tsitsiklis, *Introduction to probability*, Second (Athena Scientific, 2008).

[33] K. Hukushima and K. Nemoto, "Exchange Monte Carlo method and application to spin glass simulations", J. Phys. Soc. Jpn. **65**, 1604 (1996).

[34] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: the all convolutional net", arXiv:1412.6806v3.

[35] M. Abadi et al., "Tensorflow: a system for large-scale machine learning", `http://tensorflow.org`, 2016.

[36] C. Fang, "Phase transitions in glassy systems via convolutional neural networks", Presented at APS March Meeting 2017.

[37] H. G. Katzgraber, S. Trebst, D. A. Huse, and M. Troyer, "Feedback-optimized parallel tempering Monte Carlo", J. Stat. Mech. P03018 (2006).

[38] M. A. Nielsen, *Neural networks and deep learning* (Determination Press, 2015).

# APPENDIX A

# BACKPROPAGATION AND STOCHASTIC GRADIENT DESCENT

## A.1 Learning in the MLP

This appendix describes in detail the mathematical basis of *stochastic gradient descent algorithm*, a generalization of the SLP update rule for the MLP that is based on finding an optimal set of parameters that minimize output error, which is quantified by an appropriate cost function $\mathsf{c}$. That is, a parameter (some weight or bias) $\theta$ is adjusted such that $\mathsf{c}$ is decreased, i.e.

$$\theta \to \theta - \eta \frac{\partial \mathsf{c}}{\partial \theta}. \tag{A-1}$$

The partial derivative may appear difficult to calculate individually in an MLP, since $\mathsf{c}$ depends directly on $y$, which has a chain of dependencies up until the $l$th layer. The *backpropagation algorithm* gives an efficient way of calculating all such partial derivatives in the *reverse* direction (coming from the output layer) and thus furnishing an update rule for the parameters. The idea is elegant from the point of view of multivariable calculus: we simply traverse the tree of chain rules on $\mathsf{c}$ and propagate the values down for use in the next calculations.

Unlike the SLP and its learning algorithm, an important prerequisite is for the cost function $\mathsf{c}$ and the activation function $\phi$ to be differentiable, as their derivatives will play part in determining the parameter updates

## A.2 Gradients and Error Signals

First, we need to write all the necessary partial derivative relations, which is straightforward. Following Nielsen [38], consider the output of a non-input layer $l$

$$a_i^{(l)}(z^{(l)}) = \phi(z_i^{(l)}). \tag{A-2}$$

Its gradient with respect to $z^{(l)}$ is a diagonal matrix that can be written as

$$\frac{\partial a_i^{(l)}}{\partial z_j^{(l)}} = \phi'(z_i^{(l)}) I_{ij}^{(l)}, \tag{A-3}$$

where $I_{ij}^{(l)}$ is the Kronecker symbol (1 if $i = j$ and 0 otherwise) with dimension lengths of $\dim a^{(l)}$. Next, the cost function $\mathsf{c} = \mathsf{c}(a^{(L)})$ explictly depends on the output of the $L$th layer, but our goal will be to calculate the quantities

$$\delta_i^{(l)} = \frac{\partial \mathsf{c}}{\partial z_i^{(l)}} \tag{A-4}$$

which can be seen as the *error signal* of the neurons in layer $l$ in gradient descent. It can be related to the derivative of $\mathsf{c}$ with respect to any parameter tensor $\theta^{(l)}$ (a weight or bias) in the layer as

$$\begin{aligned} \frac{\partial \mathsf{c}}{\partial \theta^{(l)}} &= \sum_i \frac{\partial \mathsf{c}}{\partial z_i^{(l)}} \frac{\partial z_i^{(l)}}{\partial \theta} \\ &= \sum_i \delta_i^{(l)} \frac{\partial z_i^{(l)}}{\partial \theta}. \end{aligned}$$

The partial derivative $\frac{\partial z_i^{(l)}}{\partial \theta}$ is straightforward to find from (1-21). For a weight, it is $a_i^{(l-1)}$, and for a bias, it simply $1$. Thus, the expressions for the weight and bias errors are

$$\frac{\partial \mathsf{C}}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} a_j^{(l-1)} \tag{A-5}$$

$$\frac{\partial \mathsf{C}}{\partial b_i^{(l)}} = \delta_i^{(l)} \tag{A-6}$$

The backpropagation calculation thus needs to efficiently calculate the error signals $\delta^{(l)}$. It first starts with the output layer: by the chain rule, the error signal in the $L$th layer is

$$\begin{aligned} \delta_i^{(L)} &= \sum_j \frac{\partial \mathsf{C}}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_i^{(L)}} \\ &= \frac{\partial \mathsf{C}}{\partial a_i^{(L)}} \phi'(z_i^{(L)}) \end{aligned} \tag{A-7}$$

With further application of the chain rule, when $l < L$,

$$\begin{aligned} \delta_i^{(l)} &= \sum_j \frac{\partial \mathsf{C}}{\partial z_j^{(l+1)}} \frac{\partial z_j^{(l+1)}}{\partial z_i^{(l)}} \\ &= \sum_{j,k} \delta_j^{(l+1)} \frac{\partial z_j^{(l+1)}}{\partial a_k^{(l)}} \frac{\partial a_k^{(l)}}{\partial z_i^{(l)}} \\ &= \phi'(z_i^{(l)}) \left( \sum_j \delta_j^{(l+1)} w_{ji}^{(l+1)} \right) \end{aligned} \tag{A-8}$$

The above expressions relates $\delta^{(l)}$ to $\delta^{(l+1)}$. Thus, we can calculate the error signal at each layer *starting from layer* $L$ and going down.

## A.3 The Backpropagation Relations

The above finally gives us the ingredients for the backpropagation algorithm with the update rules

$$w_{ij}^{(l)} \to w_{ij}^{(l)} + \Delta w_{ij}^{(l)} \tag{A-9}$$

$$b_j^{(l)} \to b_j^{(l)} + \Delta b_j^{(l)}. \tag{A-10}$$

Where

$$\Delta w_{ij}^{(l)} = -\eta \frac{\partial \mathsf{C}}{\partial w_{ij}^{(l)}} = -\eta \delta_i^{(l)} x_j \tag{A-11}$$

$$\Delta b_j^{(l)} = -\eta \frac{\partial \mathsf{C}}{\partial b_i^{(l)}} = -\eta \delta_i^{(l)}. \tag{A-12}$$

using the equations (A-7) and (A-8) for the deltas. Algorithm 2 thus follows.

# APPENDIX B

# MARKOV CHAINS AND DETAILED BALANCE

## B.1 Definitions

We outline some basic definitions in Markov Chain theory [29, 30, 32].

**Definition 2.** *Suppose that $p(s \to s')$ is the transition matrix of a Markov Chain. Then...*

1. *A state $s'$ is said to be* accessible *from a state $s$ if the Markov chain can eventually reach $s'$ from $s$. That is $p^n(s \to s') > 0$ for some $n$.*

2. *A subset $A \subset \Omega$ is called a* recurrent class *if for every pair $s, s' \in A$, $s'$ is accessible from $s$, and no state in $\Omega - A$ is accessible from $A$.*

3. *A recurrent class $A$ is said to be* aperiodic *if there exists an integer $n$ such that $p^n(s \to s') > 0$ for all $s, s' \in A$.*

4. *If a Markov Chain has a single recurrent class $A$, and $A$ is aperiodic, the Markov Chain is said to be* ergodic

## B.2 Detailed Balance

**Theorem 2** (Global Balance)**.** *Suppose a Markov chain with a transition matrix $p(s \to s')$ is ergodic. Then it has a stationary distribution $\pi(x)$ if and only if for every state $s$,*

$$\sum_{s' \neq s} \pi(s) p(s' \to s) = \sum_{s' \neq s} \pi(s') p(s' \to s). \tag{B-1}$$

*Proof.* Rewrite (2-3) and (2-4) by singling out $s = s'$,

$$p(s \to s) = 1 - \sum_{s' \neq s} p(s' \to s),$$

and

$$\pi(s) = \pi(s)p(s \to s) + \sum_{s' \neq s} \pi(s')p(s' \to s).$$

Thus,

$$\pi(s) = \pi(s)\left(1 - \sum_{s' \neq s} p(s' \to s)\right) + \sum_{s' \neq s} \pi(s')p(s' \to s)$$

and $\pi(s)$ can be canceled to give

$$\sum_{s' \neq s} \pi(s)p(s' \to s) = \sum_{s' \neq s} \pi(s')p(s' \to s).$$

The reverse reasoning through the above three equations is also clear, so that (2-4) follows. Thus, $\pi(s)$ must be the stationary distribution of the Markov Chain by uniqueness.

□

Theorem 2 is used as motivation for a key choice behind the Metropolis-Hastings algorithm, which is based on a Markov Chain that satisfies the condition of *detailed balance* as follows:

**Corollary 1** (Detailed Balance)**.** *For a Markov Chain with an aperiodic recurrent class to have a specific stationary distribution $\pi(s)$, it is a sufficient condition for the transition probabilities to satisfy*

$$\pi(s)p(s \to s') = \pi(s')p(s' \to s) \tag{B-2}$$

*for all states $s$ and $s'$. Such a Markov Chain is said to satisfy* detailed balance*, or be* reversible*.*

Satisfying detailed balance is thus a stronger condition than satisfying global balance.

**Corollary 2.** *For a Markov Chain, specified by transition, proposal, and acceptance prob-*

*abilities $p(s \to s') = g(s \to s')A(s \to s')$, to satisfy detailed balance with a stationary distribution $\pi(s)$, it is a sufficient condition for the proposal and acceptance probabilities to satisfy*

$$A(s \to s') = \min\left(1, \frac{\pi(s')g(s' \to s)}{\pi(s)g(s \to s')}\right). \tag{B-3}$$

*Therefore, Algorithm 6 returns an estimate of an observable mean $\langle f \rangle$ through a Markov chain.*

**Corollary 3** (Metropolis-Hastings Algorithm)**.** *Suppose $\pi(s)$ is Boltzmann distributed, with $\pi(s) = e^{-\beta E(s)}/Z$, and that the proposal probability matrix $g$ is* symmetric. *Then with the acceptance probabilities*

$$A(s \to s') = \min\left(1, e^{-\beta \Delta E}\right) \tag{B-4}$$

*where $\Delta E = E(s') - E(s)$ is the energy change in the transition, the transition matrix $p(s \to s') = g(s \to s')A(s \to s')$ implements a Markov Chain Monte Carlo algorithm.*